Study and Performance Evaluation of Transport Layer Protocols over Heterogeneous Radio and Satellite Networks

by

Carl Thomas Vatne
Lars-Rune Haugen

Thesis in partial fulfilment of the degree of
Master of Technology
In
Information and Communication Technology

University of Genoa
Faculty of Engineering,
DIST

Agder University College
Faculty of Engineering
and Science

Genoa, Italy

May 2006

Abstract

Satellites have for some years made it possible to transfer data over long distances, and from desolated locations. Due to their natural broadcast ability they have some major advantages compared to terrestrial links. This ability has made satellites a useful tool for military, science, multimedia and wireless communication.

Compared to a few years ago, we are today able to make more accurate measurements, in different environments. Sensors have recent years become more intelligent, and can work independently without being controlled by humans or computers. Other improvements are their mobility, and ability to work wireless.

In recent years there has been a lot of research addressed to combining satellite communication and wireless sensors in ad hoc networks. If these two wireless technologies can be incorporated in the same network, it would result in a lot of possibilities. Scenarios where desolated wireless sensors measures independently, and sends data through a satellite link, to a gathering center are now possible. ZigBee Alliance has developed a protocol stack architecture, specialized for small wireless sensors working in ad hoc structured networks. IEEE has developed a standard for the two lowest OSI-layers, IEEE 802.15.4, which the ZigBee layers work upon.

This thesis will study and evaluate the performance of different TCP versions throughout heterogeneous wireless networks. Today is TCP NewReno the most common TCP protocol, but we will also consider other TCP versions, like TCP Westwood+, TCP Hybla and TCP Vegas. The most important aspect of this evaluation is the average throughput. To test the performance of the different TCP versions over the heterogeneous wireless links, we tested over a real satellite link, and a virtual wireless channel. We emulated transmission from a sensor node sending to a sink, through this channel. There are different ways to simulate this channel. We have looked at two models; Characteristic Model and the Two State Markov Model. The latter option is a model which uses four different parameters to define the channel. These parameters are bandwidth, delay, alpha and beta. Alpha and beta defines the transition probability between a good state (perfect transmission) and a bad state (no transmission). The first model, the Characteristic Model, uses IEEE 802.15.4 parameters and a waypoint mobility model to calculate the probability of packet loss. We have extended the already existing ACE Emulator, to emulate an ad hoc network. For our testing we decided to use the Two State Markov Model.

Based on the results we achieved, we can conclude that TCP Vegas performed best over the satellite link. We tested TCP NewReno, TCP Westwood+, TCP Hybla and TCP Vegas for this link. Unexpectedly performed TCP NewReno second best in this test. Over the virtual wireless channel performed TCP NewReno best. We tested TCP NewReno and TCP Westwood+ over this channel. Our overall conclusion is that TCP NewReno performs best in general throughout our heterogeneous wireless networks. We have to take into account the fact that we experienced some problems with both networks during testing, but the results give a strong indication of TCP NewRenos quality and range of use.

Preface

This thesis is the final stage of the Master degree in Information and Communication Technology at Agder University College, faculty of Engineering and Science. The project is given by the University of Genoa, and the work was carried out between January 2006 and May 2006 in Genoa, Italy.

We want to give a special thank to our supervisor, Ing. Tomaso de Cola, for his first-class guidance and valuable counselling throughout the project period. We will also thank our main supervisor, Prof. Franco Davoli, and our supervisor at Agder University College; Prof. Dr.-Ing. Frank Reichert, for starting the cooperation and making this project possible.

Genoa, May 2006

Carl Thomas Vatne and Lars-Rune Haugen

Table of Contents

List of Figures

List of Graphs

List of Tables

Acronyms and Abbreviations

ACK	TCP Acknowledgment Segments
AP	Access Point
APS	Application Support Sub-layer
BSS	Basic Service Set
BPSK	Binary Phase Shift Keying
BER	Bit Error Rate
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
CA	Congestion Avoidance
CWND	Congestion Window
dBm	Decibel Milliwatt
BDP	Delay*Bandwidth Product
DSSS	Direct Sequence Spread Spectrum
DNS	Domain Name System
D-SACK	Duplicate-SACK
EU	Elaboration Unit
BWE	Estimated Bandwidth
ECN	Explicit Congestion Notification
FTP	File Transfer Protocol
FEC	Forward Error Correction
GEO	Geostationary Orbit
HTTP	Hyper Text Transfer Protocol
ISM bands	Industrial, Scientific and Medical frequency bands
CNIT	Italian National Institute for Telecommunications
LLC	Logical Link Control
LEO	Low Earth Orbit
MAC	Media Access Control
MAS	Medium Access Sub-Layer
MEO	Medium Earth Orbit
Mbps	Mega bit per second
mW	Milliwatt
NWK	Network Layer
O-QPSK	Offset-Quadrature Phase Shift Keying
OSI	Open Systems Interconnection
OS	Operating System
PEP	Performance Enhancing Proxy
PSK	Phase Shift Keying
PSD	Power Spectral Density

P_{be}	Probability of bit error
P_{pd}	Probability of packet drop
PDU	Protocol Data Unit
RTP	Real-Time Transport Protocol
RWND	Receiver's Advertised Window
RTT	Round-Trip Time
SPS	Satellite Protocol Stack
STL	Satellite Transport Layer
STP	Satellite Transport Protocol
SEGSIZE	Segment Size
SACK	Selective Acknowledgement
SMSS	Sender Maximum Segment Size
SIP	Session Initiation Protocol
SNR	Signal to Noise Ratio
SMTP	Simple Mail Transfer Protocol
SS	Slow Start
SCPS	Space Communication Protocol Standard
SSTRESH	Start Threshold Size
SCTP	Stream Control Transmission Protocol
APL	The Application Layer
TCP	Transmission Controll Protocoll
UDP	User Datagram Protocol
VoIP	Voice over IP
WLAN	Wireless Local Area Network
WPAN	Wireless Personal Area Network
XTP	Xpress Transport Protocol
ZDO	ZigBee Device Object

1 Introduction

This chapter gives an overview and the layout of our thesis. The first section describes the background of the project. Section 1.2 describes our problem area, and shortly presenting the test beds and our tasks. In section 1.3 is the thesis definition presented. The last section, section 1.4, gives an overview of the report.

1.1 Background

The Transmission Control Protocol (TCP) [1] is used by popular internet traffic, as Internet browsing, file transfer and e-mail, and is therefore by far the most used Transport Layer protocol. TCP is an end to end protocol which delivers reliable data transfer. In other words, it sends information from one user to another without considering intermediate routers and, as long as a link exists between the two users, TCP guarantees that the information will arrive. To be able to guarantee reliable data transfer TCP uses, among other functions, the so called Congestion Control mechanism. This mechanism controls the rate at which data is sent in to the network and retransmits segments if necessary. If the network enters a state of congestion (that is, the buffers in the routers or receiver are overloaded and segments are dropped) the Congestion Control mechanism reduces the sending rate. Segment drops are noticed at the sender side when acknowledge packets fail to arrive. This mechanism is fundamental for the Internet to work properly. But, in some cases, Congestion Control has an unwanted effect. Every segment drop is treated as congestion in the network, even though congestion may not be the cause. Some wireless systems are error prone and segment drops may be caused by corruption. When corruption occurs on a link, TCP makes an unwanted reduction of the sending rate. This causes low utilization of the bandwidth, which already is a scarce resource in some wireless systems. This problem is enhanced in networks with long delays and large Round Trip Time (RTT).

Different variants of the TCP protocol have been suggested, and modifications have been evaluated to optimize TCP for error prone networks. The most recognized variants of TCP in general are TCP NewReno [2] and TCP Westwood+ [3]. Other protocols have also been suggested, for instance TCP Vegas [4], TCP-Peach [5] and TCP Hybla [6]. A lot of work has been done to mitigate the unwanted effects of TCP in error prone networks; some of them are described in [7, 8, 9, 10, 11].

TCP NewReno is derived from the basic TCP Reno, and uses the four Congestion Control algorithms: Slow Start, Congestion Avoidance, Fast Retransmit and Fast Recovery [2]. The Slow Start algorithm is used initially of a session, and when time out occurs. Slow Start increases the sending rate exponential, up to a given point. After that, the Congestion Avoidance algorithm is invoked. In the contrary, this algorithm increases the transmission rate linearly, to avoid reaching a state of congestion too fast. Congestion Avoidance continues until congestion is detected. The Fast Retransmit algorithm is invoked when congestion occurs and packets are lost. The transmission rate is reduced, and after the lost packet are retransmitted, the Fast Recovery will try to catch up with the former transmission rate by invoking Congestion Avoidance again.

TCP Westwood+ has modified the Fast Recovery algorithm; with this modification the name has been changed to Faster Recovery [3]. Instead of halving the sending rate after congestion is detected, the Faster Recovery algorithm tries to measure the actual available bandwidth and then adjust the data rate accordingly.

Many modifications and mitigations have also been suggested to avoid the unwanted effect of TCP in error prone networks. A highly discussed modification is the implementation of Selective Acknowledgements (SACK). This mechanism overcomes the problem of multiple packet drops by sending a SACK message, which contain ACKs for all the received data segments [8]. The receiver is able to select and retransmit only the lost segments. Another discussed subject is Path MTU Discovery. This mechanism tries to use all the available bandwidth by sending the largest possible segment size [11]. Also Performance Enhancing Proxies (PEPs) have been proposed as mitigation. PEPs are proxies that can store and retransmit segments [9], they can also change the transport protocol and parameters for the different links in the transmission path. This reduces the retransmission time for data segments. PEPs are usually implemented at the application layer, but can enhance performance in other layers as well. The problem with PEPs is that they interfere with the end to end transmission.

It is essential that modifications and mitigations are friendly towards each other and the current implementation; that is, they can work together in the same network without causing trouble and negative effects for each other. For instance, a new TCP modification must be friendly towards the current implemented TCP version. It is also important that the modifications and mitigations perform well on a variety of different networks.

As mentioned, a lot of research has been done to evaluate TCP in error prone networks. However, tests of TCP in multiple wireless networks have not been numerous, and especially not with the Zigbee, which is a relatively new and interesting technology. Zigbee has emerged as the most attractive standard for wireless sensor systems. A Zigbee sensor network consists of low power ad hoc devices, and supports different topologies. To be able to use the sensor network in desolated and unreachable places, it may be desirable to combine the sensor network with satellite networks.

1.2 Problem Area

Our work is to evaluate different TCP versions over a heterogeneous wireless network. This network will consist of two wireless technologies, a satellite networks and an ad hoc network. This corresponds to a scenario where a remote and desolated ad hoc sensor network is connected to a computing centre over a satellite network. Ad hoc and Satellite networks have the common characteristics of error prone networks, but differ greatly in other aspects, for instance packet the sizes and the throughput performance. Because of the differences, we assume a PEP is used between the networks. Therefore we will evaluate the two networks separately.

In the evaluation of satellite networks we will use a genuine satellite test bed, while the ad hoc network will be emulated. It is our task to do the necessary changes to an already existing satellite emulator so we are able run ad hoc emulations.

Our task is to evaluate some of the most common TCP variants for wireless networks. For the satellite network we will evaluate four TCP variants: TCP NewReno, TCP Westwood+, TCP Vegas and TCP Hybla. TCP NewReno is always interesting to look at and compare to, since it is the most common implementation of TCP. TCP Westwood+ is a similar and also a widely recognized version of TCP. The last two variants, TCP Vegas and TCP Hybla, are specialized for long delay error prone networks. For the ad hoc network we will use a wider scenario, and because of time constrains, we will only evaluate TCP NewReno and TCP Westwood+. The evaluation will be based upon the throughput. Other quality of service parameters and security is not within the scope of this project. The evaluation will only take in to account a single connection over a dedicated link, which means that we will not evaluate friendliness or performance in a link with different traffic patterns.

Transport Layer protocols over error prone and long delay networks is a widely discussed subject. Prior evaluations of TCP over satellite have been done in [12,13]. A lot of works have also been done in the subject of ad hoc networks, however they are mostly simulations, one of these is [14].

1.3 Thesis Definition

The title of the thesis is:

"Study and Performance Evaluation of Transport Layer Protocols over Heterogeneous Radio and Satellite Networks"

The final definition of the work is the following:

"*Wireless networks, especially satellite networks, differ greatly from terrestrial networks in the sense of long Round-Trip Time (RTT) and Bit Error Rate (BER). Nevertheless these technologies are combined in networks today, and use the same transport protocols, i.e. Transmission Control Protocol (TCP). One of TCP's characteristics is that it assumes all packet losses to be caused by congestion in the network, and will therefore respond by invoking congestion control and avoidance algorithms. This is not always the best solution. Wireless networks are unstable, and packet loss may be caused by a non-congestion-related reason like bit errors, handoffs and variance in bandwidth. The main problem behind the project title is to find an existing protocol that copes with the characteristics of the different wireless network technologies. The protocol needs to be able to handle long delays, error prone networks and must be friendly towards current TCP implementation. In this thesis project we will compare different protocols over wireless heterogeneous networks. This means that we have a network consisting of different wireless technologies. The different technologies are ZigBee, Satellite link and Wireless Local Area Network (WLAN) 802.11. All these three networks are based on Internet Protocol (IP). ZigBee is a relatively new technology, as it was released in 2003. The problem of finding a reliable transport protocol that works over different technologies is a genuine problem, as networks like ZigBee and Satellite links become more common to combine. To decide which transport protocol is the best overall, we have to test and compare them in different scenarios. The transport protocols to compare are mainly different versions of TCP, like Westwood+, NewReno, and Sack.*"

1.4 Report Outline

The second chapter gives general information about the two networks we will evaluate, satellite and wireless ad hoc sensor networks. Chapter 3 describes the different Transport Layer protocols, TCP and UDP, and different versions of TCP. The next chapter describes the test beds for the satellite network and the ad hoc network, including the physical structure and software. Chapter 5 describes the two models for emulating the wireless channel, the Characteristic Model and the Two State Markov Model. Chapter 6 presents the different parameters, with argumentations, for the two scenarios we tested. Chapter 7 describes the results we experienced, and discussion why these results were achieved. The final conclusion and suggestions for further work are given in chapter 8.

2 Networks to Evaluate

We will evaluate Transport Layer protocols over two different network technologies: a satellite network and an ad hoc sensor network. Figure 1 shows an overview of the networks. For our evaluation, the sensor nodes are senders, while the satellite modem in the far end of the satellite network is the receiver. This corresponds to communication from left to right in Figure 1. Both networks are wireless, error prone networks. However, they have some major differences in other aspects. The nodes in a wireless sensor ad hoc network are low-power units with limited buffer size. For Transport Layer protocols which require acknowledgements, nodes with small buffer size commonly transmit with small segment sizes to be able to have a reasonable amount of segments inflight. That is, to have a reasonable amount of segments not acknowledged for. Satellite networks on the other hand have a lager buffer capacity.

Consider an example where sender A transfer a 1Mb file to receiver B. If the segment size is low, for instance a segment with 100 bytes payload (100 + 20 = 120 bytes including header), A have to send 1000000/100 = 10000 segments for the file to arrive (assuming perfect conditions). By increasing the segment payload to 1440, only 1000000/1440 ≈ 700 segments have to be sent. This increase in segment size will drastically reduce the amount of data which have to sent, because 9.300 less headers are sent. More header data will reduce the throughput of the useful information. The ad hoc sensor nodes are restricted to send with a small segment size. If this small segment size also were used in the satellite network, sub optimal throughput would be experienced in this network. The solution is to use a PEP (Performance Enhancing Proxy) between the two networks. The PEP stores segments until a reasonable large number of segments are received. Then it resizes and forwards the segments to the satellite modem which sends them over the satellite link. Because the satellite link is much faster then the ad hoc links, the segment will be transferred in bursts. The PEP is also able to change between different implementations of the TCP protocols. This way the PEP can optimize the transmission over each link.

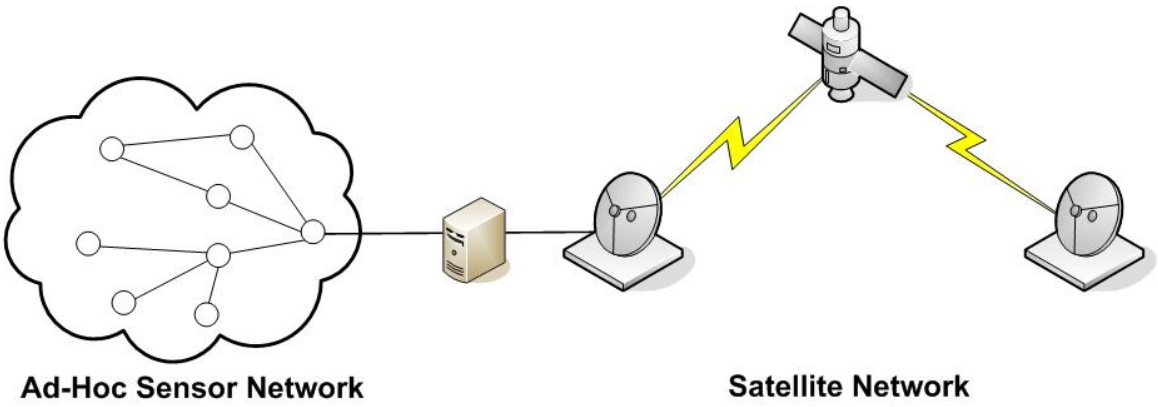


Figure 1 : Network Overview

The next sections describe the networks in detail. The last section describes some lower layer mitigations for wireless, error prone networks. PEPs are only considered for the Transport Layer, and are therefore described in section 3.1.8.

2.1 Satellite Networks

Satellites are used for a variety of purposes, like television broadcast, radio broadcast, Internet communication and positioning systems. They are often characterized by the orbit they are located in and by what their purpose is. This work will focus on communication satellites, which may be located in Low Earth Orbit (LEO), Medium Earth Orbit (MEO or ICO) or Geostationary Orbit (GEO).

2.1.1 GEO Satellites

Satellites in GEO are located at an altitude of approximately 36,000 km above the equator. At this altitude the satellite orbits the earth at the same speed as the earth rotation period. Thus, as seen from the earth the satellite will always have the same position in the sky. For all satellite networks, the ground based antennas must be directed towards the satellite before the two parts can communicate. Since GEO satellites always have the same position in the sky, ground based antennas always know where to direct their signals. Thus, the complexity of the base station is decreased, because tracking systems are not necessary (in contrast to MEO and LEO satellites). For communication with many base stations, this can more than justify the extra cost of onboard complexity and sending the satellite to higher altitudes. Only three GEO satellites are needed to cover the surface of the earth (poles are not included). Because of the high altitude, the propagation delay for a signal traversing a GEO satellite network is considerable. The propagation time for a signal traveling twice that distance is 239.6 milliseconds [7]. This corresponds to a signal traveling from an earth station to a satellite and back to an earth station, when both earth stations are directly beneath the satellite, i.e. a best performance scenario. For the opposite, when both earth stations are at the edge of the satellites view, the signal has to travel 41,756 km * 2 [7], which would means a propagation delay of 279,0 milliseconds. If we take into account round trip time (RTT) in the latter scenario, we will experience a propagation delay of 558 milliseconds. The RTT corresponds to traffic from one earth station to satellite and to another earth station, and then back again the same route. The delay is also dependent on other factors, such as switching and buffering. Experience shows that GEO satellites are more reliable and have a longer operation lifetime then MEO and LEO satellites. On the other hand, a GEO satellite has a wider region of operation and if it experiences a malfunction or for some reason is unreachable, it is very critical because it affects more users.

2.1.2 MEO Satellites

MEO satellites are located in orbit at an altitude of about 10,000 km. Because MEO is closer to earth then GEO, more satellites are required to achieve the equal amount of coverage. A GEO satellite can be in view of a base station for several hours. To be able to offer reliable and optimal services, a MEO constellation often consist of between 10 and 17 satellites, orbiting in two or three orbital planes [15]. Each base station has to implement tracking systems, which are able to locate satellites and perform handovers. Despite this, MEO are often preferred instead of GEO because of the shorter propagation delay.

2.1.3 LEO Satellites

LEO satellites are stationed in orbit about 1000 km above the surface of the earth. These satellites use less then two hours to circle the earth. This means that they will be in sight of a base station for about 20 minutes. Thus, the ground equipment must be constantly ready for handovers. To reduce timeout periods LEO satellite constellations are meant to have more then one satellite in view at any place, at any time all over the world. This requires complex tracking procedures and handover mechanisms. The LEO satellites have two major advantages compared to GEO, and also MEO; these are the short propagation delay and the ability to communicate with small devices. The propagation delay varies from a few milliseconds to 80 milliseconds, depending on the distance from the base station to the satellite [7]. Because the signal travels a relative short distance, mobile handheld devices are able to pick up the signal sent by a LEO satellite. This widens the satellites range of services, and introduces many interesting possibilities.

2.1.4 Link Characteristics

The link characteristics of satellite communications can be described by three main properties; delay (described earlier), noise and bandwidth.

The strength of the signal degrades in proportion to the square of the distance it travels [7]. In satellite networks the distance between sender and receiver can be vast, thus the signal will be considerably weakened before arrival. This means that there will be a low signal-to-noise ratio upon arrival. Weather conditions can have a negative effect on the link; especially rain is degrading for the performance of satellite links. In communication from satellite to handheld mobile devices, the link is especially susceptible to multi-path distortion and shadowing, (i.e. the signal is blocked by tall buildings)[7]. Error control coding schemes, like Forward Error Correction (FEC), can be implemented to increase signal to noise ratio. Even though the satellite links are considered error prone, a typical bit error rate (BER) for satellite links today is 1 error per 10 million bits (1 * 10^-7) or less.

The radio spectrum is a limited resource, which is restricted by licenses. The bandwidth is separated in different bands, where the most commonly used bands for communications satellites are: Ku (Kurz-under) band, Ka (Kurz-above) and the C (Compromise) band. The Ku band ranges 10-17 GHz. The Ka band ranges 8-31 GHz [16], where the 20/30 band is used by communications satellites. The C band ranges from 4 to 6 GHz.

Satellites links can be both symmetric and asymmetric. For symmetric communications, forward and return link have the same bandwidth limitations, while for asymmetric communications the limitations may differ. In many situations symmetric communication is not bandwidth efficient. This could for instance be the situation where the return path has less bandwidth requirement, like ACK traffic. Another reason for implementing asymmetric communications is because of limits on the transmission power and the antenna size at one end of the link (e.g. mobile phones)[10]. For these reasons asymmetric communication is widely used in satellite networks.

It should also be noted that satellites have a natural broadcast capability. Satellites are superior compared to terrestrial links for this type of communication.

2.1.5 OSI Layers

Only two OSI layers are defined for satellite networks, the Physical Layer and the Data Link Layer. Figure 2 shows a typical satellite network. The modems can be stand-alone units, as in the figure, or they can be implemented in a router with a modem card. With the latter solution the Data Link Layer will communicate directly with the routers' Network Layer.

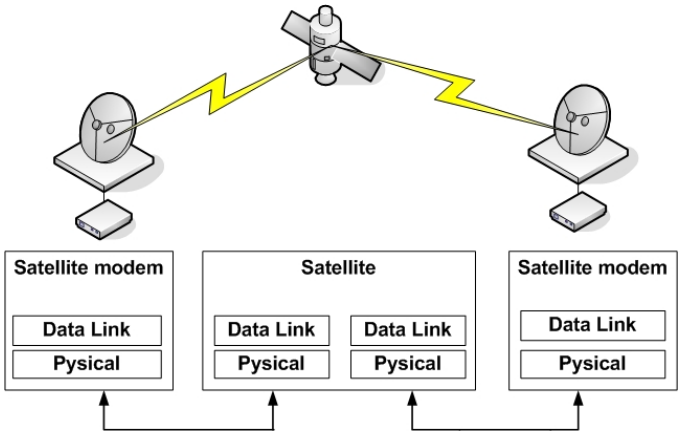


Figure 2 : OSI Layers in Satellite Networks

The responsibilities of the Physical Layer is to transmit streams of bits from the transmitter to the receiver with minimal bit errors, for both broadcast and unicast transmissions. This involves the following tasks: selection of the frequency band, generation of carrier frequency, modulation and last: transmission and detection of transmitted bits. The task of the Link Layer is to encode and decode data packets into bits. It handles flow control and frame synchronization. The Link Layer can be divided in to two sublayers; the Medium Access Control (MAC) and the Logical Link Control Layer (LLC). The MAC sublayer is responsible for channel access and for permission to transmit on the channel. The LLC layer controls frame synchronization, flow control and error checking.

It should be noted that the Data Link layer in the satellite uses a form of switching. This switching routes the traffic to the different satellite transponders, and should not be mixed with routing at the Network layer, which is commonly performed on terrestrial routers.

2.2 Wireless Ad Hoc Sensor Networks

Zigbee [18] is the second standard in this project. In the Zigbee architecture the Physical and Link Layers use the IEEE 802.15.4 standard for Wireless Personal Area Network (WPAN), while higher layers are defined by the Zigbee specification. The nodes are low powered wireless sensors connected together in a network. The connections are peer-to-peer associations based upon the ad hoc principle. The next sections describe this in detail.

2.2.1 Wireless Sensor Networks

In an ad hoc network, the nodes communicate directly peer-to-peer, without going through a hub, switch or router. The nodes act as both router and terminal at the same time. An ad hoc network is also called an Independent Basic Service Set (IBSS), which is the simplest IEEE 802.11 network, since no network infrastructure is required. Wireless sensor networks are often used for monitoring purpose and

measurements, where the object to be observed can vary from short-range to wide range. A short-range measurement object can potentially be temperature monitoring, and wide-range can be environmental surveillance. Some of the application areas are health, military, and home. In military, for example, the rapid deployment, self-organization, and fault tolerance characteristics of sensor networks make them a very promising sensing technique for military command, control, communications, computing, intelligence, surveillance, reconnaissance, and targeting systems [19]. These tiny sensor nodes consist of sensing, data processing, and communicating components. Instead of sending the raw data to the nodes responsible for the fusion, they use their processing abilities to locally carry out simple computations and transmit only the required and partially processed data. This makes sensor networks represent a significant improvement over traditional sensors.

Although these sensors vary in range of applications, they all suffer from the same constraints:
Density: High-end microsensor networks are expected to have a density of approximately 20 nodes/m3. Hence, the medium access control layer (MAC) should be able to accommodate several hundreds to thousands of nodes.
Distributed traffic: Due to their high node density, wireless sensor networks must have a high capacity. However, the data rate requirements per node are low (<10 kbps). This results in a very low radio duty cycle.
Energy: Microsensornodes are required to be small and autonomous. Their small form factor limits the amount of energy that can be stored in batteries. Furthermore, the density of the network as well as the environment where nodes are deployed often prohibits periodic replacement of the batteries. An existing goal is for a microsensor node to have an average power on the order of 100mW, which would allow the device to obtain its power from the environment by energy scavenging. [15]

There are discussed ways to improve the matter of constraints, but this is not an issue in this report.

2.2.2 Wireless Ad Hoc Networks

A wireless ad hoc network is a collection of autonomous nodes or terminals that communicate with one another by forming a multihop radio network and maintaining connectivity in a decentralized manner. Since the nodes communicate over wireless links, they have to contend with the effects of radio communication, such as noise, fading, and interference. Each node in a wireless ad hoc network functions as both a host and a router, and the control of the network is distributed among the nodes [7]. The network topology is in general dynamic, because the connectivity among the nodes may vary with time due to node departures, new node arrivals, and the possibility of having mobile nodes. The Internet Engineering Task Force Mobile Ad hoc Network Work Group, IETF MANET WG, are now developing and testing different routing protocols for wireless ad hoc networks, to define a standard protocol. The protocol has to support both dynamic and static topologies, and both IPv4 and IPv6. The scenarios used in this research are IP-structured networks with wireless ah-hoc networks at the edges, with both fixed and mobile routers. This work is estimated to be finished during 2006 [17].

2.2.3 Zigbee Layers

ZigBee is a relatively new technology, and has since 2005 offered a low-cost, very low power consumption, two-way, wireless communications standard. It also needs less than a second to initiate a connection, which makes it ideal to use in sensor networks to send small amounts of data.

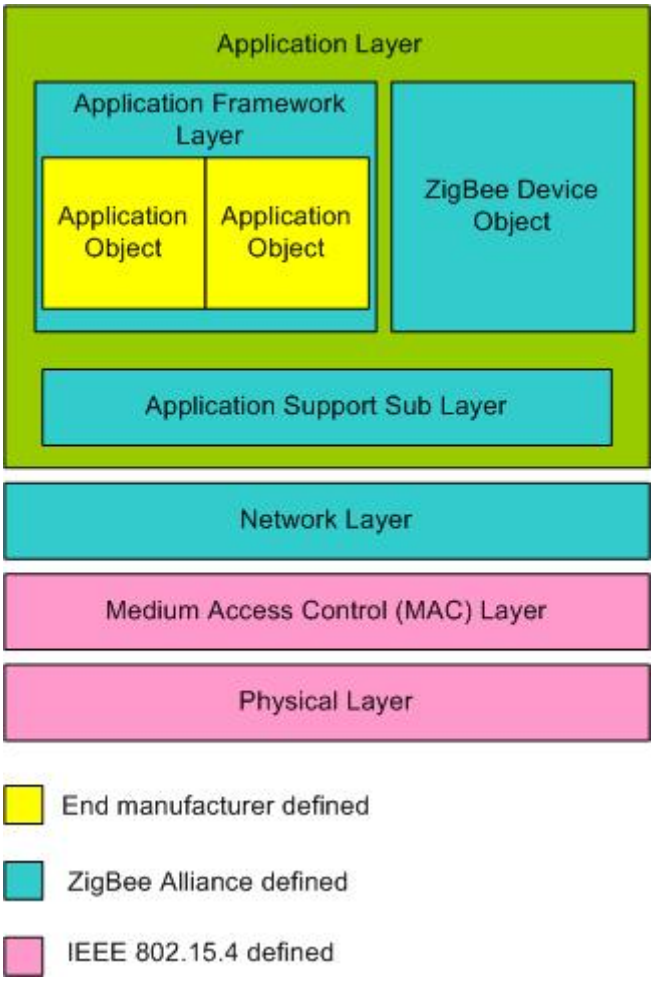


Figure 3 : ZigBee and IEEE 802.15.4 Layers

The ZigBee stack architecture is based on the standard Open Systems Interconnection (OSI) seven-layer model, with some modifications. As Figure 3 shows, has the ZigBee Alliance defined the upper layers. While the two lower layers; MAC and Physical layer, are defined by the IEEE 802.15.4. The ZigBee layers are described below, and the IEEE 802.15.4 layers are described in section 2.2.4.

The application layer (APL) is the upper ZigBee Alliance layer. It consists of three sub-layers: the Application Support Sub-layer (APS), the ZigBee Device Object (ZDO) and the manufacture-defined application objects [18]. The Application Objects can vary a lot, depending on the purpose of the ZigBee node.

The responsibilities of the APS sub-layer include maintaining tables for binding, which is the ability to match two devices together based on their services and their needs, and forwarding messages between bound devices. This layer corresponds to the OSI layer four, the transport layer. It adds an APS-header to the pdu, and pushes it down to the network layer. The responsibilities of the ZDO are to define the role of the device within the network, e.g., ZigBee coordinator or end device, discovering devices on the network and determining which application services they provide, initiating and responding to binding requests and establishing a secure relationship between network devices. The APS provides an interface between the Network layer (NWK) and the Application layer (APL) through a general set of services that are used by both the ZDO and the manufacturer-defined application objects.

The responsibilities of the ZigBee Network layer (NWK) layer shall include mechanisms used to join and leave a network, to apply security to frames and to route frames to their intended destinations [18]. It covers also discovery and maintenance of

the routes between devices, finding new one-hop neighbours and storing pertinent neighbour information. The NWK layer of a ZigBee coordinator is responsible for starting a new network, configuring new devices, and assigning addresses to newly associated devices. The ZigBee network layer supports star, tree and mesh topologies.

2.2.4 IEEE 802.15.4 Layers

The goal for the IEEE 802.15.4 group was to develop an architecture with very low complexity, typically for low data rate, multi month to multi year battery life, and which can operate in an unlicensed, international frequency band. Typical applications can be home automation, wireless sensors, interactive toys, smart badges and remote controls [20]. Two layers have been developed for this architecture, they are described in the following sections.

The Medium Access sub-layer (MAC) is responsible for transmitting beacon frames, synchronization, and providing a reliable transmission mechanism [18]. It also controls access to the radio channel by using a Carrier Sense Multiple Access – Collision Avoidance (CSMA-CA) mechanism. CSMA/CA works as follows: A station wishing to send, first senses the medium. If the medium is idle, the sender waits a prescribed minimum of time after the last transmission sensed before it sends, to reduce the chance that a message transmitted by another station has not yet been sensed by the one in question. If the medium is sensed busy, the sender waits a random time (more than a minimum of time), before it senses the medium again. This procedure reduces the possibility that two or more stations are trying to transmit at the same time [21].

IEEE 802.15.4 has two physical layers, which operate on two distinct frequency ranges; 868/915 MHz and 2.4 GHz. The lower frequency range, 868/915 MHz, is used in Europe (868 MHz) and United States and Australia (915 MHz), respectively. The higher range is used worldwide [18]. Typical data rates are 250 kbps in the 2.4 GHz band, 40 kbps in the 915 MHz band, and 20 kbps in the 868 MHz band. [14] Both physical layers use Direct Sequence Spread Spectrum, DSSS. The 2.4 GHz physical layer uses Offset Quadrature Phase-Shift Keying modulation, O-QPSK, while 868/915 MHz physical layer uses Binary Phase-Shift Keying modulation, BPSK [22].

2.3 Lower Layer Mitigations for Error Prone Networks

Errors due to bit corruption are a big problem for TCP communications, where all packet loss is treated as network congestion. If a TCP sender detects packet loss, it reduces its sending rate, even though the packet loss is caused by corruption and not congestion. This results in an unwanted reduction of sending rate, and is an especially bad effect for networks with long delay. A detailed description of the behaviour of TCP is given in chapter 3. Many modifications have been proposed to mitigate the constraints in long delay and error prone networks. This chapter will discus some of these mitigations, with respect to the Physical, Data Link and Network Layers.

Bit Error Rate (BER) is a parameter which describes the number of bit errors. The BER is affected by many parameters, including physical and environmental factors as well as signal processing parameters. Some physical and environmental factors are

propagation delay, scattering, noise and signal power. The unwanted effects of physical and environmental factors cannot easily be mitigated from a physical point of view. However, by increasing the transmission power the signal power (useful information) will be stronger compared to the unwanted background information (noise). This ratio is known at signal to noise ratio (SNR). When SNR is large, the received signal is equal to the transmitted signal, and when SNR is low the signal may be modified so much that wrong information is received. When the signal power is increased, the SNR will also increase. In some networks the nodes are energy self-sufficient or run on batteries, thus energy is a scarce resource. In these networks it is desirable to achieve acceptable BER through other means. With respect to signal processing, high BER can be mitigated through modulation and error correction schemes, by sacrificing bandwidth for decreased BER. In modulation, a pattern of bits is always defined, these patterns are called symbols. How many bits a symbol contains may vary for each type of modulation scheme. A demodulator maps the symbols, that is, it analyses the symbols according to a diagram, which can be seen as a co-ordinate system. If a symbol consists of many bits, the distance between the symbols is small and the signal becomes less resilient to errors. If the symbol consists of just a few bits, the distance among the symbols is larger and the signal is more resilient to errors. For error prone networks it is common to implement error detection and corrections schemes. The most common implemented scheme at the physical layer is Forward Error Correction (FEC). FEC adds redundant data to the transmitted information, by using a predefined algorithm. Many different algorithms have been proposed for FEC. How much data the algorithms add depends on the level of resilience wanted. By adding more redundant data, the signal becomes more resilient to errors. An example of FEC is to transmit the same bit three times, so the transmitted bits are 000 instead of just 0, and 111 instead of just 1. The receiver will check each series of three bits, and see which bit value occurs most. All the values 000, 001, 010 and 100 have most 0's, thus are treated as 0. The values 011, 101, 110 and 111 have most 1's and are treaded as 1. For this level of FEC, one bit in a series of three (not necessary each third) can be corrupted without causing any bad effects for the communication. This example of FEC is very simple and also very inefficient since three times as much data have to be transmitted.

Another way to avoid unwanted reduction of the sending rate is to add awareness of corruption. If a router experiences corruption above a given threshold, it sends a new "corruption experienced" ICMP message to the TCP receiver [10]. The TCP receiver informs the TCP sender of corruption through an option in the ACK message. When the TCP sender receives a "corruption experienced" message, it assumes all packet loss is due to corruption for the duration of 2 RTT or until it receives additional link state information. By ignoring segment loss for two RTT, this proposed mechanism might aggravate congestion.

A lot of research also focuses on making the data link layer detect errors and perform data link layer retransmission. The main problem with this solution is that the segments are not received in the same order as they were sent. For instance, assume a series of 5 segments are sent, segment 1, 2, 3, 4 and 5. If segment 2 has to be retransmitted somewhere along the path, the receiver might receive segment 2 last, and the order will be 1, 3, 4, 5, 2. For segment 3, 4 and 5 duplicate ACKs will be created and TCP will invoke the fast retransmission. This is not necessary, as the next segment to arrive is the one which appears to be missing. One solution is to suppress

or delay duplicate ACK in the reverse direction. Another proposed solution is to make the TCP more capable of handling out-of-order segments [10].

Another proposed mitigation is Path Maximum Transmission Unit (MTU) Discovery. This mechanism detects the largest datagram size that is possible to send over the entire path, without being fragmented. This is the most efficient datagram size, and is called Path MTU [11]. If a host sends a datagram with size less then Path MTU, it will probably experience suboptimal throughput. If the host uses a datagram size larger then Path MTU, some routers will fragment the package. This fragmentation uses Internet resources, and should be avoided. The Path MTU Discovery works as follows; the sender sets the "Don't Fragment" (DF) flag when a datagram is transmitted. If this datagram is received by a router whose MTU is smaller than the size of the packet, an ICMP "Datagram too Big" message is sent back to the sender. This ICMP message indicates that the message is too big to forward, and contains the MTU size for the router. The sender adjusts the MTU size and resends the packet. This process is repeated until the packet can traverse the whole path without being fragmented.

Another widely discussed topic is Performance Enhancing Proxies (PEPs). PEPs can be implemented at any layer, but are usually implemented on the Transport layer or Application layer. PEPs are therefore described in section 3.1.5.

3 Transport Layer Protocols

The transport layer is the fourth layer in the OSI model. The most common layer four protocols are Transmission Control Protocols (TCP), User Datagram Protocol (UDP), Real-Time Transport Protocol (RTP) and Stream Control Transmission Protocol (SCTP). TCP is used for reliable data traffic, like file transfer (File Transfer Protocol, FTP), e-mail (Simple Mail Transfer Protocol, SMTP) and web (HyperText Transfer Protocol, HTTP). TCP is described in section 3.1. UDP does not offer the same reliability as TCP, but it delivers faster. Common applications that use UDP are Domain Name System (DNS), Voice over IP (VoIP) and online gaming. UDP is described in chapter 3.2. RTP carries voice and video data when having videoconferences over the Internet. Session Initiation Protocol (SIP) is used for setting up and tearing down these conversations. SCTP is an extension of TCP, capable of handling several streams at the same time. SCTP is another transport protocol used for voice activity over the Internet.

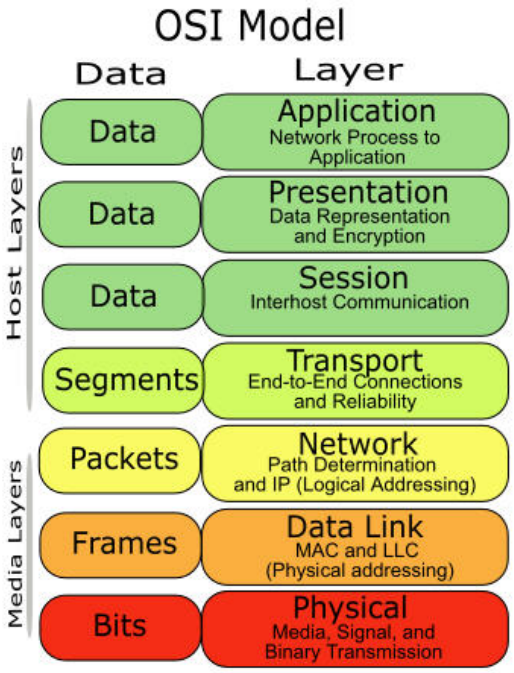


Figure 4 : OSI Model

TCP is the most important transport layer protocol for this project, and it is described in details in the section 3.1. The subsequent sections describe the most common TCP versions and improvements. The following section outlines research directly related to satellite networks, including new TCP versions and improvements.

3.1 Transmission Control Protocol

The great advantage with TCP is its reliability to deliver accurate data, and consideration for other transmitters within the same network. To begin with the reliability, TCP starts every session with a three-way-handshake. Consider TCP communication between part A and part B, as shown in Figure 5. First source A sends a SYN segment to destination B, to ask if B is ready to start the TCP session. If B is ready, it replies with a SYN ACK segment. A replies with an ACK segment to acknowledge the SYN ACK from B, and then the session transmission can start. To keep track of which segments have arrived properly, each segment in a TCP session is given a sequence number.

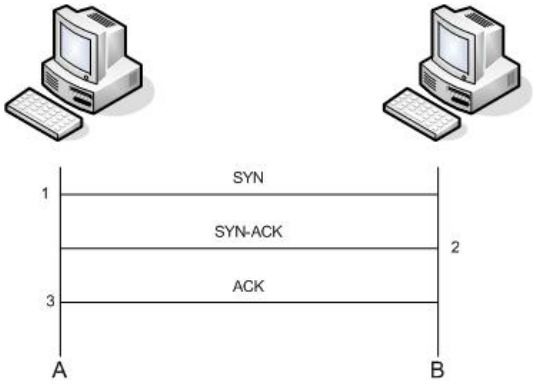


Figure 5 : Three-Way-Handshake

In real transmission the sequence number is the number of sent bytes, but for simplicity numbered with simple numbers in this example. When A sends segments, they are numbered e.g. 1001, 1002, 1003 and 1004 respectively. B will then acknowledge with the number of the next sequence number it expects to receive. E.g., when B receives segment 1002, it will acknowledge with sequence number 1003. If every segment is received properly and acknowledge for, then TCP sender A will increase sending rate. When A increases the sending rate, there may be segments 'floating' that are not acknowledged for, before A sends new segments. If one of the segments is lost during the transmission, or by other reasons not received properly, B will continue to acknowledge the last received segment. E.g., if sender A sends a bulk of segments with sequence number from 1003 to 1007, and segment number 1005 is lost during transmission, then B will acknowledge segment nr 1003 and 1004, even if it has received segment nr 1006 and 1007 as well. When receiver B receives segment nr 1006 and 1007 but not 1005, it will send a duplicate ACK for 1004, for each segment received over 1005. With this behavior from B, sender A will know that segment nr 1005 was lost during transmission, and retransmit segment nr 1005. Then B will acknowledge segment 1005, 1006 and 1007 respectively.

To make sure every arriving segment is correct, TCP uses a 16 bit checksum in the TCP header. This checksum field contains one's complement of the one's complement sum of all 16 bit words in the header and text [23]. If a segment contains an odd number of header and text octets to be checksummed, the last octet is padded on the right with zeros to form a 16 bit word for checksum purposes. The pad is not transmitted as part of the segment. While computing the checksum, the checksum field itself is replaced with zeros. A segment with incorrect checksum will be discarded, and retransmitted.

The TCP protocol was initially designed to work in networks with low link error rates, i.e., all segment losses were mostly due to network congestions. Radio links suffer from some different challenges than terrestrial links, and in recent years there has been a lot of research with main focus to error prone- and long delay networks. To fully utilize the bandwidth, satellite links needs to have a lot of data "in-flight"[1]. This is because satellites links have long delays and a relatively large bandwidth. This is characterized by the so-called Delay * Bandwidth Product (DBP). Improvements for the Transport Layer are mostly some modifications of the TCP versions and modifications of the TCP/IP semantics, called Performance Enhancement Proxies (PEP).

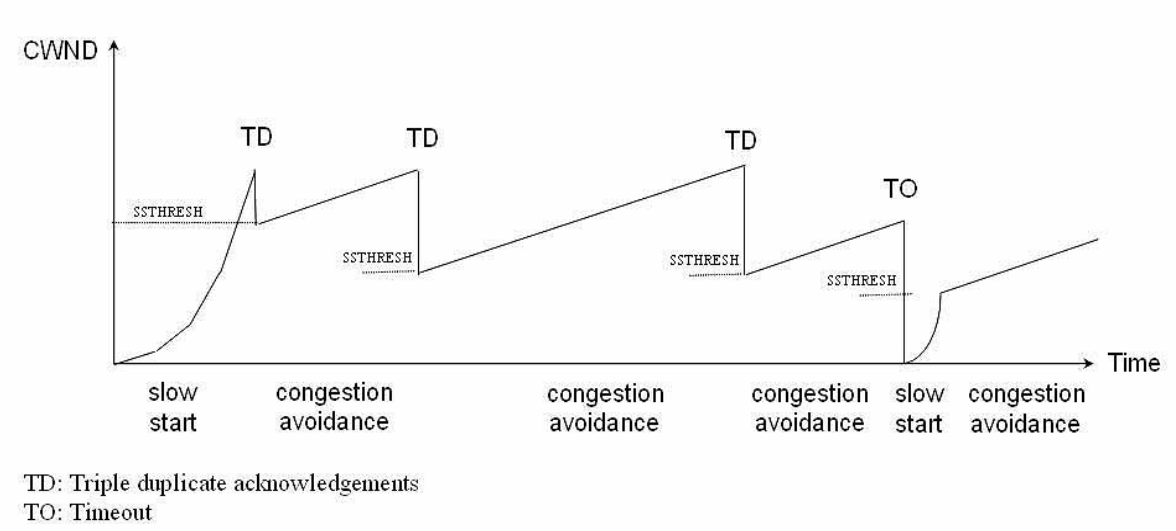
The following sections describe some important TCP versions, including versions in which we will evaluate. Section 3.1.1 to 3.1.5 describes different general TCP versions. The next sections, 3.1.6 to 3.1.8, describe TCP versions specialized for the satellite network, and Performance Enhancement Proxies.

3.1.1 TCP Reno

For the consideration for other transmitters, and avoid congestion in the network, TCP uses Congestion Control. Congestion Control contains four algorithms; Slow Start (SS), Congestion Avoidance (CA), Fast Retransmit and Fast Recovery [1]. These are

[1] That is, a lot of data which is in the network, but not acknowledged for.

the four basic algorithms for TCP, and this basic TCP-version is called RENO. When sender A starts a transmission, it does not know anything of the networks conditions. To decide the transmission rate, two variables have to be considered: the Congestion Window (CWND) and Receivers Advertised Window (RWND). CWND is a sender-side limit on the amount of data the sender can transmit into the network before receiving an ACK. RWND is a receiver-side limit on the amount of outstanding data. The minimum of these two variables governs the data transmission. By using SS, the sender initially sets the CWND to one segment and sends one segment, and then it waits for the ACK. When the ACK is received, the sender doubles the amount of segments to send and adjusts the CWND, to two segments. When ACK for these two segments is received, it doubles the CWND and sends four segments, and so on. Another important variable for Congestion Control is the Slow Start Threshold Size (SSTRHESH). The initial value of SSHTRESH is 65535 bytes [24]. SSTRESH is



Graph 1 : CWND and SSTHRESH for TCP Reno

used to determine whether SS or CA is used to control the data transmission. The SS algorithm is used when a transmission session is initialized, and when CWND < SSTRHESH. The CA algorithm is used when CWND > SSTRHESH. When CWND and SSTRHESH are equal, the sender may use either SS or CA [23]. When congestion occurs, indicated by timeout, the SSTHRESH value will be set to the half of CWND and the CWND is set to one, as shown in Graph 1. SS will now start over again. When CWND again reaches SSTHRESH, CA will take over and continue. CA increases the transmission rate linearly, in contrast to SS's exponential growth. For every ACK the sender receives, it increases the transmission rate by one segment each Round-Trip-Time (RTT) [24]. When a sender receives a duplicate ACK, it does not know if the segments have arrived out-of-order or have been lost during transmission. To decide, it waits to receive more duplicate ACKs. If it is just a segment out-of-order, it will only receive one or two duplicate ACKs, before the segments are reordered. But if the sender receives three or more duplicate ACKs in a row, it assumes that the segment has been lost [24]. TCP then performs a retransmission of the missing segment without waiting for the retransmission timeout to expire. This algorithm is called Fast Retransmission. After Fast Retransmit has finished, Fast Recovery will be performed. In short words Fast Recover invokes CA instead of SS (as shown in Graph 1), because one lost segment does not necessary mean that the

network is congested. Fast Recovery uses a new variable, Sender Maximum Segment Size (SMSS). Fast Retransmit and Fast Recovery are usually implemented together, and work as follows [1]:

1) When the third duplicate ACK has arrived, the SSHRESH is set to:

   $$SSHRESH = \max (FLIGHTSIZE^2 / 2, 2 * SMSS)$$

2) Retransmitt the lost segment, and set CWND to SSHRESH plus 3*SMSS. The 3 extra segments are for the three buffered segments at the receiver side, which have already left the network.
3) For each additional incoming duplicate ACK, the CWND are incremented by SMSS. This inflates the congestion window in order to reflect the additional segment that has left the network.
4) Transmit a new segment, if allowed by the new value of CWND and RWND.
5) When ACK for the new segment has arrived, the CWND is set to SSTHRESH. This ACK also acknowledges all the intermediate segments sent between the lost segment and the receipt of the third duplicate ACK.

By using Fast Recovery, TCP can continue with a higher throughput even though it had to retransmit a segment.

One of the disadvantages with TCP is the transmission time. Every transmission session is initiated by the three-way-handshake, which means that three segments have to be sent before actual data can be sent. In some cases, like DNS, this is a waste of time. Another disadvantage is the way TCP acts in the presence of long delay. There may be a long delay in wireless networks, especially Satellite networks, and the retransmission time can timeout. The long delay can simply be a result of a long distance, and the segments may arrive perfectly in just a moment. But TCP will anyway act as if congestion has occurred, reduce the transmission rate and start to retransmit. This is a totally illogical and unwanted way to handle the problem.

There has been a lot of research to cope with TCPs reaction to the delay problem, but also to improve TCP in general. General improvements will be described first, and improvements especially for error prone- and long delay networks will be described in the subsequent sections.

3.1.2 TCP SACK

One improvement for handling multiple segment loss is the Selective Acknowledgement Option (SACK). With cumulative acknowledgement, the sender has to wait one RTT to find out about each lost segment. This is a waste of time and an ineffective way to decide which segments it has to retransmit, especially if several segments are lost. An aggressive sender may retransmit more segments than it has to because it doesn't bother to wait for all the duplicate ACKs. If the sender supports SACK, a "SACK-permitted"-flag is set in the SYN-packet. If supported, the receiver chooses if it wants to use this option during the session. The SACK-option divides the

[2] FLIGHTSIZE is the amount of outstanding data in the network.

incoming segments into blocks, where each block represents received segments that are contiguous and isolated. The left edge of the block is the first sequence number of this block, and the right edge of the block is the sequence number immediately following the last sequence number of this block [8]. This means that the sequence number just below and above the block has not been received, as shown in Figure 6.

Figure 6 : Overview of contiguous segment blocks

For simplicity are the segments given simple numbers, instead of number of sent bytes, as in real transmission. The SACK-acknowledgment is sent by the receiver to inform the sender that non-contiguous blocks have arrived and been queued. Here are segments number 1004 and 1005 missing. The TCP receiver sends ACK for segments number three, indicating it is ready to receive segment number four, as normal. But if segment number six is received, but not number four, as in this case, will the receiver send a SACK acknowledgment. This SACK contains ACK for segment number three, indicating segment number four is missing, and ACK for segment number six. This informs the TCP sender that it has to retransmit segment number four and five. If the next incoming segment is number seven, will the SACK contain ACK for segment number three, six and seven respectively. But if the incoming segment is number four, the SACK contains ACK for segment number four and six. The right edge of the block is now moved one segment to the right. This way the sender knows exactly which segments have been lost, and retransmits just the right segments. This algorithm can decrease the time TCP spends to choose which segments to retransmit, and makes TCP more effective.

A SACK option that specifies n blocks, will have the length of 8*n+2 bytes. [8] This means that the 40 bytes available for TCP options can specify maximum 4 blocks. But SACK is often used together with the Timestamp option used for RTT. This options uses 10 bytes, thus each SACK can contain a maximum of 3 blocks.

SACK-segments acknowledges lost packets in a more informative way than normal acknowledgements, witch makes the retransmission more effective. However, sometimes are packets retransmitted by a mistake, or by other reasons sent more than once, even if it was properly received. This behavior is less effective, and totally unnecessary. Duplicate-Sack (D-SACK) [34] is an extension of SACK, and handles this problem. D-SACK reports duplicate contiguous sequences in blocks, like SACK, and make use of the first block in SACK. The left edge of the D-SACK block specifies the first sequence number of the duplicate contiguous sequence, and the right edge of the D-SACK block specifies the sequence number immediately following the last sequence in the duplicate contiguous sequence. Since D-SACK is an extension of SACK, can it not be used unless the SACK-option is used.

3.1.3 TCP NewReno

SACK may not be used for every connection, as it is not always supported locally or a TCP peer does not want to use the option. In cases of multiple segments loss when

using the basic Reno, the TCP sender lacks information when deciding which segments to send during Fast Recovery. When the sender receives three duplicate ACKs, it concludes a segment loss and retransmits the lost segment, indicated by the duplicate ACKs. If there are more segments in flight when the sender enters the Fast Retransmit, the sender may receive more duplicate ACKs. The first new information available for the sender, is the ACK for the retransmitted segment in Fast Retransmit. If there was a single segment drop, and no reordering, this ACK will acknowledge all the segments transmitted before Fast Retransmit was entered. However, if there was a multiple segment loss, then this ACK will acknowledge some but not all the segments transmitted before the Fast Retransmit was entered. This is called Partial Acknowledgement. A new version of Reno, called NewReno, has modified the Fast Retransmit and Fast Recovery algorithms to improve TCP when multiple segments are lost. NewReno makes use of a new variable, 'recover', whose initial value is the initial send sequence number [2]. The NewReno Fast Retransmit and Fast Recovery work as follows [2]:

1) If the sender is not already in the Fast Recovery procedure when it receives three duplicate ACKs, it checks if the Cumulative Acknowledgement field covers more than 'recover'. If so, go to step 1a. Otherwise, go to step 1b.
    a. Invoke Fast Retransmit, and SSTHRESH is set to:

        SSTHRESH = max (FLIGHTSIZE / 2, 2 * SMSS)

        The highest sequence number transmitted is recorded in 'recover'. Go to step 2.
    b. Do not enter the Fast Retransmit and Fast Recovery procedure, do not go to step 2 to retransmit the lost segment, and do not execute step 3 upon subsequent duplicate ACKs. TCP will continue to send traffic and increase the CWND as normal.
2) Enter the Fast Retransmit and retransmit the lost segment. CWND is set to SSTHRESH plus 3*SMSS. The 3 extra segments are for the three buffered segments at the receiver side, which have already left the network
3) Enter the Fast Recovery. For every additional duplicate ACK received while Fast Recovery, the CWND is incremented by SMSS. This inflates the congestion window in order to reflect the additional segment that has left the network.
4) Continue Fast Recovery and transmit a segment, if allowed by the new value of CWND and RWND.
5) When an ACK arrives that acknowledges new data, this ACK could be the acknowledgement elicited by the retransmission in step 2, or elicited by a later retransmission.

    Full acknowledgement:

    This means the ACK acknowledges all the intermediate segments sent between the original transmission of the lost segment and the receipt of the third duplicate ACK. The CWND is set to either A) SSTHRESH, where SSTHRESH is the value set in step 1) to deflate the window, or B) min(SSTHRESH, FLIGHTSIZE + SMSS). (Notice that FLIGHTSIZE in step 5 refers to the amount of outstanding data in step 5, when Fast Recovery is exited. FLIGHTSIZE in step 1, on the contrary refers to the amount of outstanding data in step 1, when Fast

Recovery was entered.) One of these two options, A) or B), is set when the TCP version is implemented, and chosen either by the Operating System or a skilled user. Exit the Fast Recovery Procedure.
Partial Acknowledgement:
The Partial Acknowledgement does not acknowledge all the data up to, and including 'recover'. The sender has to retransmit the first uancknowldged segment. Deflate the CWND by the amount of new data acknowledged by the Cumulative Acknowledgment field. If the Partial ACK acknowledges at least one SMSS of new data, then add back SMSS bytes to the CWND. As in step 3, it has to take into account if additional segments already have left the network too. It permitted by the new value of CWND, a new segment is transmitted. This mechanism ensures that when Fast Recovery ends, approximately SSTHRESH amount of data will be outstanding in the network.

The retransmit timer is reset when the first partial ACK arrives during Fast Recovery.

6) If the retransmit times out, the highest sequence number transmitted is stored in 'recover'. Exit Fast Recovery.

There are two variations of NewReno; the Impatient variant and the Slow-but-Steady variant. With the Impatient variant the retransmit timer is reset only after the first partial ACK. In case of many dropped segments in the same window, the sender's retransmit timer will ultimately expire, and TCP will invoke Slow-Start. On the contrary, the Slow-but-Steady variant will reset the retransmit timer after each partial ACK. In case many dropped segments in the same window here, the sender will retransmit at most one segment per RTT. Studies show that Reno performs better than NewReno in the presence of reordering, while NewReno is superior in the presence of multiple segment loss. Even though Reno performs better when segments are reordered, it is recommended to use NewReno in general, because its superiority when handling multiple segment loss outweighs its poorer performance when handling reordering. The optimal option to use is SACK, in both cases of reordering and multiple segment loss.

3.1.4 TCP Westwood+

Another well-known version of TCP is Westwood+. TCP Westwood+ is an enhanced version of Westwood. Westwood works as Reno, but instead of Fast Recovery Westwood uses Faster Recovery. In contrary to Fast Recovery, Faster Recovery takes bandwidth-estimation into account when setting the SSTHRESH. To do so, Faster Recovery introduces a new variable; Estimated Bandwidth (BWE). BWE is calculated for every incoming ACK, but it has implemented a procedure that considers delayed ACK, duplicate ACK and selective ACK for the calculation to be more accurate [12]. Westwood works as follows:

1) When ACK is received, the BWE is calculated and CWND is increased by one, accordingly to the Reno algorithm.
2) If three duplicate ACKs are received:
   SSTHRESH = (BWE * RTT_{min}) / SMSS
   CWND = SSTHRESH

3) When a coarse time out expires:
   SSTHRESH = (BWE * RTT_{min}) / SMSS
   CWND = 1

To make the calculation of BWE more accurate, Westwood+ makes the calculation of BWE every RTT in contrary to Westwood. Experiments infer this enhancement to be very successful [12].

3.1.5 TCP Vegas

TCP Vegas is based on the same Congestion Control as Reno, but has improved the Retransmission, Congestion Avoidance and Slow Start algorithms. TCP Reno is waiting for congestion to occur with segments loss and then act with Congestion Avoidance. In the contrary TCP Vegas calculates a difference between estimated output and measured output to avoid congestion. To do this, TCP Vegas introduce a new variable; BaseRTT. Each time the TCP Vegas sender sends a segments, it reads and records the system clock. When an ACK arrives, Vegas reads the clock again and calculates the actual RTT. BaseRTT is the minimum of all measured round trip times. This way, the TCP Vegas use a more accurate RTT than TCP Reno. The new Retransmit algorithm has become more effective by using the accurate RTT, and works as follows:

> *In case of duplicate ACK:* TCP Vegas checks if the difference between the current time and the timestamp recorded for the relevant segment is greater than the timeout value (default around 500ms [4]). If it is, the sender will retransmit the missing segment without waiting for three or more duplicate ACKs.
> *In case of non-duplicate ACK:* If this ACK is the first or second after retransmission, Vegas has to check if the time interval since the segment was sent is larger than the timeout value. If so, the segment has to be retransmitted. This will catch any segment that has been lost previous to the retransmission without having to wait for a duplicate ACK.

This way TCP Vegas will decrease the CWND only if the retransmitted segment was previously sent after the last decrease. A loss that happened before the last CWND decrease should not affect current CWND, to make it decrease one more time. This is an important improvement from TCP Reno.

To decide if the CWND should increase or decrease during congestion avoidance, TCP Vegas compares the outputs. Its goal is to maintain the ultimate available bandwidth, and not cause any segment loss. First it has to calculate an expected throughput [4]:

Expected = CWND / BaseRTT

Then it has to calculate the actual throughput. This is done by recording the sending time for a segment, record how many bytes (B) are transmitted between the segments was sent and its ACK is received and calculating the RTT.

Actual = B / RTT [4]

The difference, Diff, is defined as follows [4]:

Diff = Expected – Actual

The difference is compared with two predefined thresholds, alpha and beta respectively. The thresholds are defined in terms of KB/S, and are numbers for how many extra buffers the connection is occupying in the network. To decide if the there is too much or too little data in the network. Alpha and beta are defined as:

alpha < beta

If Diff < alpha, Vegas increases the CWND linearly the next RTT, and decreases the CWND linearly if Diff > beta. The CWND is unchanged if alpha < Diff < beta.

The slow start procedure is modified by implementing the new congestion detection mechanism from CA, to avoid segments loss. During the SS, the CWND is just exponential increased every other RTT. In between the CWND stays unchanged, for the comparison of expected and actual throughput. If the actual rate falls below the expected rate by a certain predefined amount, Vegas change from SS mode to linear increase/decrease mode. During SS TCP Reno sends two segments for every ACK, in the contrary TCP Vegas sends only as much data as is acknowledged for.

3.1.6 TCP Peach

TCP Peach is a new TCP version, developed for wireless networks with long delay. It is composed of two new algorithms: Sudden Start and Rapid Recovery, as well as the traditional algorithms Congestion Avoidance and Fast Retransmit. Sudden Start and Rapid Recovery replace the Slow Start and Fast Recovery algorithms, respectively. TCP Peach uses the unusual concept of dummy-segments to probe the availability of network resources. The dummy-segments do not carry any new information to the sender, and are therefore low-priority segments and will be dropped first in case of congestion. Due to the low-priority, the segments do not cause any decrease of data throughput of actual data [6]. The dummy-segments have one or more of the unused flags set in the TCP-header, the responding ACKs have the same flags set [5]. If the TCP receiver does not support dummy-segments, the TCP Peach sender stops sending dummy-segments and start to behave like TCP Reno. When Slow Start algorithm is used in a long delay network, it has to spend a lot of time to increase the throughput. Sudden Start increases throughput much faster while using dummy-segments; at the beginning of a connection, the CWND is set to 1, and after the first data segment is sent, the sender transmits (RWND-1) dummy-segments for every (RTT / RWND) [5]. As a result, after one RTT, the CWND size increases very quickly. The sender can estimate RTT during the connection setup phase. Rapid Recovery is invoked when the Fast Retransmit is completed, and lasts for one RTT. For every received ACK, two dummy-segments are sent in order to evaluate the available bandwidth. Congestion Avoidance is invokes after one RTT. The acknowledgements for the dummy-segments sent during the Rapid Recovery phase cause a fast increase of the CWND. This way the channel bandwidth is utilized more effectively.

3.1.7 TCP Hybla

TCP Hybla is developed for networks with long RTT, especially Satellite networks, and its goal is a transmission rate independent of the actual RTT. To do so, TCP Hybla has implemented a set of procedures to enhance the already existing set of rules for Slow Start and Congestion Avoidance in Congestion Control. The new rules requires some new variables; CWND(t), B(t) and P. CWND(t) is the congestion window evaluated in time, expressed in SMSS. B(t) is defined as the segment transmission rate, expressed as the amount of segments pr second [6]:

$$B(t) = CWND(t) / RTT$$

P is the normalized round trip time, and it is defined as the ratio between the actual RTT and the round trip time of the reference connection to which TCP Hybla aims to equalize the performance, denoted by RTT_0.

$$P = RTT / RTT_0$$

The new set of rules for SS and CA is represented as follows:

$$W_{i+1} = \begin{cases} W_i + 2^{\rho} - 1 & SS \\ W_i + \dfrac{\rho^2}{W_i} & CA \end{cases}$$

These new rules make the CWND increase much faster. It is important to increase the RWND as well, to avoid limiting the transmission rate. Multiple losses in the same window will be more frequent due to the increase of CWND. To avoid unnecessary retransmission TCP Hybla takes advantage of the SACK option. These modifications make TCP Hybla Congestion Control much more effective than the TCP standard [6].

3.1.8 Performance Enhancing Proxies

PEPs are used to enhance the performance over links with problematic characteristics. They are usually implemented at Application layer or Transport layer, but may also functions on the other layers as well. A quality of PEP, is their ability to operate transparently to the other systems involved, like end systems, transport endpoints and applications. PEP implementations are said to be either integrated or distributed. Integrated is when i.e. it comprises a single PEP component in a single node. Distributed is said to be two or more PEP components typically in multiple nodes. A distributed implementation is often used to surround a particular link, like satellite link, for which performance enhancement is required.

The two most important types of PEPs are those based on Spoofing and Splitting. Spoofing is a way to hide the long delay at the Satellite link, by making the gateway transmit a ‘fake-ACK’ to the TCP sender, for every segment. In this way the transmission rate will increase much faster than normal. The main drawback with Spoofing is that the end-to-end semantics is no longer respected. In addition, the gateway has to intercept the ‘true-ACK’ sent to the TCP sender, to avoid duplicate

ACK. If a segment is lost, the gateway will be responsible for the retransmission of the missing segment. Retransmission can in this case be faster, because the retransmitted segment does not have to be retransmitted from the sender. The PEPs act as TCP senders and receivers, and it is important both the segment and the acknowledgement pass through the same PEPs.

Splitting means to split the connection into three separate connections; two connections between the terrestrial links to the satellite gateways, and the third connection between the gateways over the satellite link. To do this, the satellite gateways architecture has to be changed. The Satellite Protocol Stack (SPS) Architecture is implemented in Relay Entities, which isolate the satellite components from the rest of the network. A new transport layer, called Satellite Transport Layer (STL), is developed for the gateways, and it implements a new protocol, called Satellite Transport Protocol (STP). This PEP will make it possible to have different TCP versions in the connections, but it will also break the end-to-end semantics. Two transport layer protocols have been developed, with the specific purpose of working between two Relay Entities. The Xpress Transport Protocol (XTP) is a transport protocol designed for the long-latency, high-loss and asymmetric bandwidth that are typical for satellite communication [6]. The other new transport protocol is the Space Communication Protocol Standard (SCPS). SCPS is based on TCP and UDP with some extensions: “TCP Extension for High Performance”, “TCP for Transactions”, “Selective Negative Acknowledgements” and “Header Compression”.

There are some end-to-end issues when implementing PEP. By implementing PEPs in the gateway, each ACK is checked to determine if the ACK is of interest, to avoid unnecessary duplicate ACKs. IPSec encrypts the IP packet, including application and transport headers, into the IPSec Encapsulation Security Payload. Due to the IPSec, it is impossible for the intermediate PEP to examine the application and transport header and the PEP may not function optimally or at all [9]. Thus the possibility of using IPSec is restricted or sometimes absent in cooperation with PEP. However, sometimes it is more desirable with a higher throughput than end-to-end security. In these cases, IPSec can be used separately between the end systems and PEPs. When PEPs are used, there is no alternative path between the end systems. Hence a failure in the intermediate node would result in a termination of the connection. [9] As mentioned before, will the use of PEP neglect the end-to-end argument, in order to achieve reliable end-to-end delivery of data [9].

Another drawback with PEP implemented above the Network layer, is the requirement of more processing power per packet. [9] Thus the PEP will always be one or more step behind the routers in terms of the total throughput they can support. PEP implementation requires memory of every connection state, and may therefore have a limit on the number of connections it supports.

3.2 User Datagram Protocol

UDP is the opposite transport layer protocol, compared to TCP. It is an unreliable service that provides no guarantees for delivery and no protection from duplication datagrams. Because UDP skips the set up phase with three-way-handshake, it is called a connectionless protocol. Whenever a UDP sender wants to send data to a receiver, it just sends and hopes the receiver is ready to receive. The UDP sender does not buffer

the datagrams it sends, so if a datagram is lost during transmission, it is not possible to retransmit it. The UDP protocol does not contain any form of information about the connection state, and will not reduce the transmission rate in case of congestion. Since UDP does not contain any mechanisms for reliable transfer, it header is just 8 bytes. In the contrary, TCP has 20 bytes header. Due to the short header, it is faster to send UPD datagrams than TCP segments. As a security mechanism, UDP has implemented the same checksum as TCP, but it is optional to use.

4 The Test bed

The test bed is treated as two different networks separated by a PEP. In one end we have the emulator, which is responsible of emulating an ad hoc wireless sensor network. The emulator is connected to a satellite network through the PEP. This is illustrated in Figure 7. As described in chapter 2, the ah-hoc devices are treated as senders and the satellite modem in the far end of the network as receiver. From a theoretical point of view the segments will be transferred from an ad hoc sensor node, through an ad hoc network and then over two satellite links. Because we decided to use a PEP between the two networks, the networks can be evaluated independently. We will first evaluate communication in the ad hoc network using an emulator, and then we will evaluate communication in the satellite network.

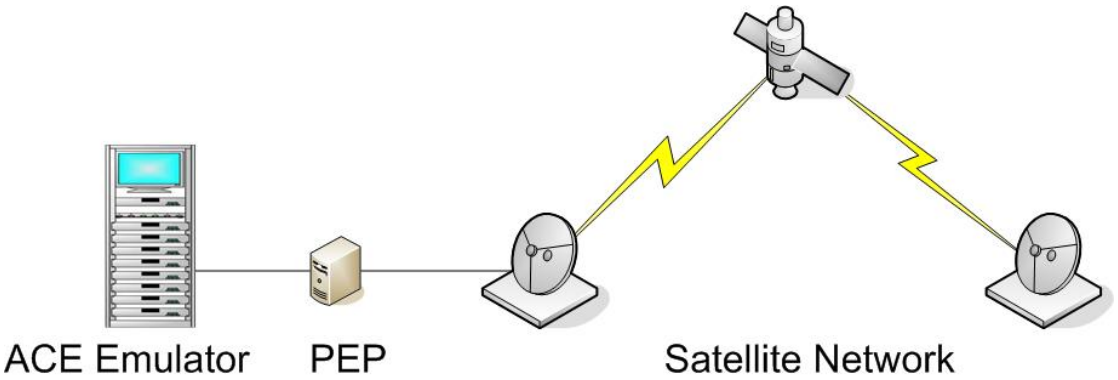


Figure 7 : The Test bed

The ad hoc network is emulated by an emulator created at the CNIT (the Italian National Consortium for Telecommunications) lab in Genoa. This emulator is called ACE (ASI CNIT Emulator), and was created for CNIT, and funded by the Italian Space Agency (ASI), with the intent to emulate satellite networks. The satellite network on the other hand, is a genuine satellite network, where one modem is stationed in Genoa and one in Naples. Section 4.1 gives some details about the satellite network, and in section 4.2 is the ACE Emulator described.

4.1 Satellite Network

CNIT provides a satellite network, which interconnects 24 different research units. The satellite network uses the Hotbird 6 satellite, which is one of 5 satellites in the Hotbird constellation. The Hotbird satellites are stationed in geosynchronous equatorial orbit at 13ºE and deliver on-board satellite multiplexing of digital television, radio and data to Europe, North Africa and most of the Middle East. The Hotbird satellites use the SKYPLEX access system developed by Eutelsat. Hotbird 6 offers a total of 32 transponders (28 Ku band and 4 Ka band) and eight SKYPLEX units for on-board multiplexing [25]. The terminal can receive a downlink stream of up to 36 Mbps user data rate from a total of 18 uplink carriers, with a combination of 6 Mbps or 2 Mbps carriers [26]. Measurements show that this satellite network has an average RTT of 701 ms, and max and min RTT of 913 ms and 557 ms [28]. The SKYPLEX terminal can operate in two modes: continuous (SCPC) or burst (TDMA) [26].

The terminal is provided by ViaSat Inc. This terminal include an IP router and a 10/100 BaseT LAN connection. We used this to connect the TCP sender (an ordinary workstation computer with Linux OS) with the terminal. Table 1 presents some details concerning the SKYPLEX system[26].

SKYPLEX Data Terminal Specification	
Uplink	
Modulation:	QPSK
Transmitt IF Frequency Range:	2150-2300 MHz
Hopping Bandwith:	150 MHz
Nominal Transmit IF signal level:	0 to -40 dBm
Transmit IF impedance:	75 ohm
Transmit return loss:	>11 dB
Downlink	
Modulation Type, spectral shaping, descrambling and FEC decoding:	ETSI EN 300 421 compliant with inner convolutional code rates only 1/2, 2/3 or 3/4
Receive Symbol Rate:	27.5 Mbaud
RF Input Frequency Range:	1350-1500 MHz
Input Power:	Desired Carrier: -60 to -30 dBm
Aggregate Power:	< -5 dBm
Input Impedance:	75 ohms
Input return loss:	-10 dB min

Table 1 : SKYPLEX Data Terminal Specification

4.2 Emulator

To evaluate the performance of new technologies, it is essential to do measurement tests. The availability of real devices makes the measurement phase easier as well as faster. However, a real platform is not always available and, moreover, is the case of wireless sensor nodes, large areas, high number of antennas would be required. As a consequence, the deployment of such a system would be expensive and time consuming, hence not feasible. Three other approaches are often used instead; analytical, simulation and emulation. The first applies theoretical frameworks to evaluate the performance. Simulation, on the other hand, tries to reprosuce a genuine system by software. Simulation is usually based on assumptions and models. For complex systems, the models are often simplified, and simulation may not have the desired degree of resemblance to the genuine system. Emulation increases the degree of resemblance, by using hardware and software together. In emulation the hardware acts like the hardware would normally act in a genuine system, and software is used to simulate other “non controllable” scenario aspects, like weather conditions and terrain properties.

The CNIT research unit at the University of Genoa has designed and implemented the ACE system to test the efficiency of on-board satellite circuit and packet switching [29]. It is our task to do the necessary changes, if any, to the ACE system, so we also can emulate ad hoc links, and adjust the parameters according to the 802.15.4 specification.

4.2.1 Physical Structure

The ACE system consists of nine computers; the Emulation Unit (EU) and eight Gateways (GTW). The task of the EU is to simulate the channel. Based upon the channel characteristics it decides if the packet is dropped. This is described in chapter 5. The EU is connected to each of the eight GTWs. Originally the GTWs were responsible for managing the specific functionalities of a satellite modem in a real system [27]. For our emulation the GTWs manage the functionalities of an ad hoc node. The architecture of the emulator is illustrated in Figure 8.

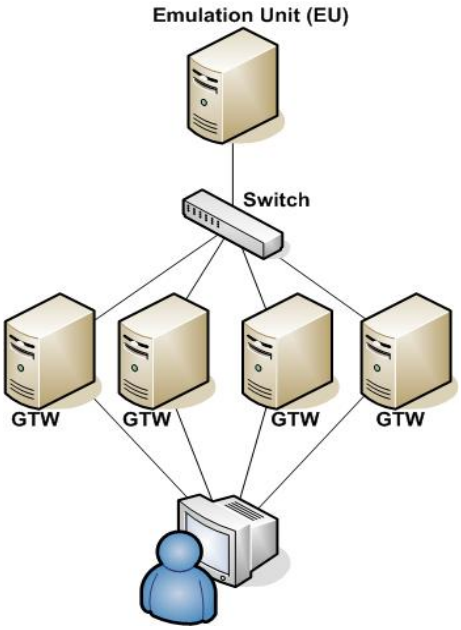


Figure 8 : Emulator Architecture

TCP sender and receiver can either be GTWs, or other computers connected to the ACE system. If external computers are connected to ACE, this is done using a 100 Mbps Ethernet link directly connected to one of the GTWs. The GTWs and the EU are connected through a closed 100 Mbps switched network. Since we only emulate long delay networks, the delays in these links are insignificant to the delay in the network emulated, thus it is disregarded. In emulation, the packets are handled differently for each emulation method, this is described in chapter 5.

4.2.2 Software

For this project, we have used different computers with different software. All software is free open source software. The ACE protocol, including Two State Markov Model and the Characteristic model, is developed by CNIT staff and us. Description of the different programs is given below. The Emulator (EU) and the Gateways (GW) run the ACE protocol. When we use the Two State Markov Model, we use external computers for TCP senders and receiver. As shown in the figure below, Adrastea is the receiver. To act as a receiver, it runs Iperf as a server. In addition it runs Tcpdump and Tcptrace to monitor and measure the throughput. Rebecca and Metis run Iperf too, but as clients. Metis is used for TCP NewReno, and Rebecca for TCP Westwood+ respectively. For the Characteristic model, the gateways will acts as sender or receivers also. The TCP receiver will in this case naturally run the same programs as the Adrastea, and the senders will run Iperf as clients.

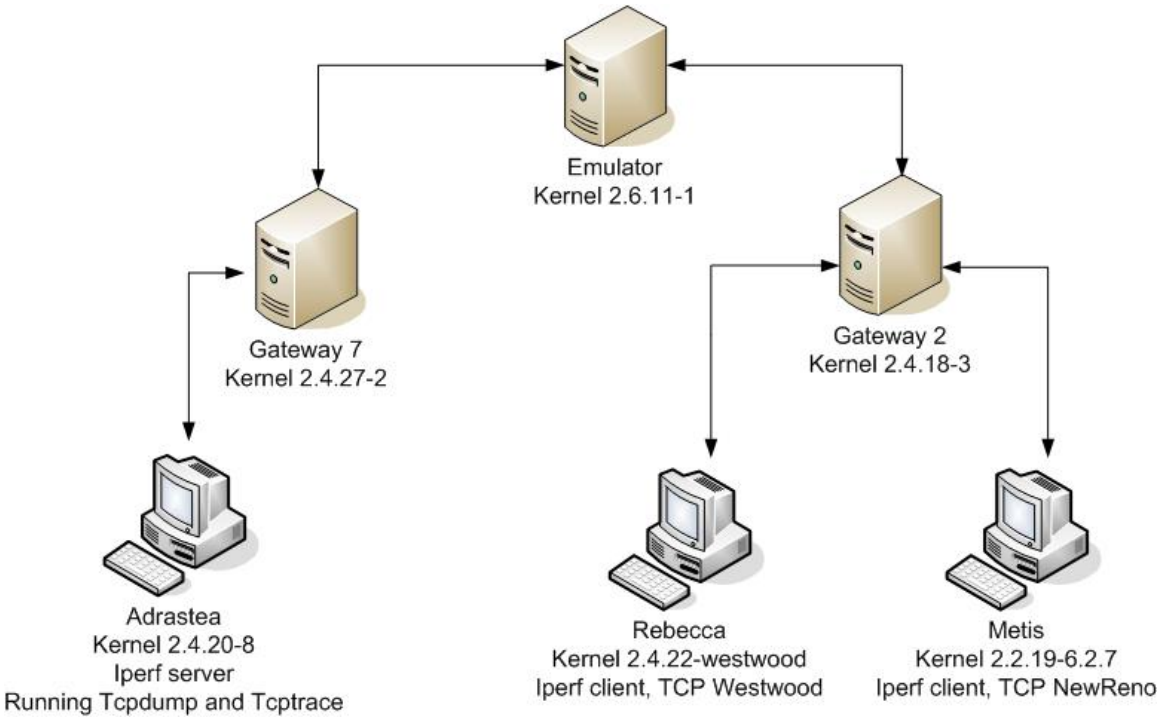


Figure 9 : Overview over Test Bed

ACE Gateways

For both the Two State Markov Model and the Characteristic model, the GWs and the EUs way to handle Packet Data Units (PDUs) and Control Packets (CP) is programmed in C/C++. These two models are described in details in section 5.1 and 5.2, respectively. In the Two State Markov Model, the TCP sender A sends a PDU to TCP receiver B. At the same time A sends a CP to the EU. The EU decides if the packet should be dropped or not, based on calculations with different parameters. This calculation is described in section 5.1. In the mean time, B has to store the packet, to wait for instruction from the EU. A new CP is sent from the EU to the receiver B, which either drops or reads the packet based on the orders from the EU. In the Characteristic Model, an additional CP is sent from receiver B to the EU. This CP contains Bs position, so the EU can calculate if the packet should be dropped or not. This calculation is described in section 5.2. The GW structure regarding the handling of packets among the layers, is equal to both models. A TAP-device is implemented in between the Data Link layer and the Network layer. By use of the TAP-device, the GWs are actually connected by means of a virtual network that takes the PDUs and transports them as if they were transmitted over a real satellite system [29]. As shown in Figure 10, PDUs travels between the GWs, and CPs between the GWs and the EU. When a PDU is arriving in the GW from an external sender, like Metis and Rebecca in the Two State Markov Model, it naturally arrives at the Physical layer (as shown with red line). The PDU is pushed upwards the layers, as in normal routers and gateways. At the network layer, the PDU is pushed downwards, this time through the TAP-device, Data Link layer and out through the Physical layer. When the PDU is arriving at the TAP-device, a CP is sent to the EU (shown with blue arrow). This packet follows the same path as the PDU, when leaving the GW. When the PDU is arriving in the receiving GW, it naturally arrives at the Physical layer and is pushed upwards. As mentioned earlier, is the PDU stored at the receiver side, pending instructions from the EU (CP illustrated with blue arrow). The PDU is stored in the

TAP-device. If the Characteristic model is used, the TAP-device will also send a CP to the EU (as shown with green arrow).

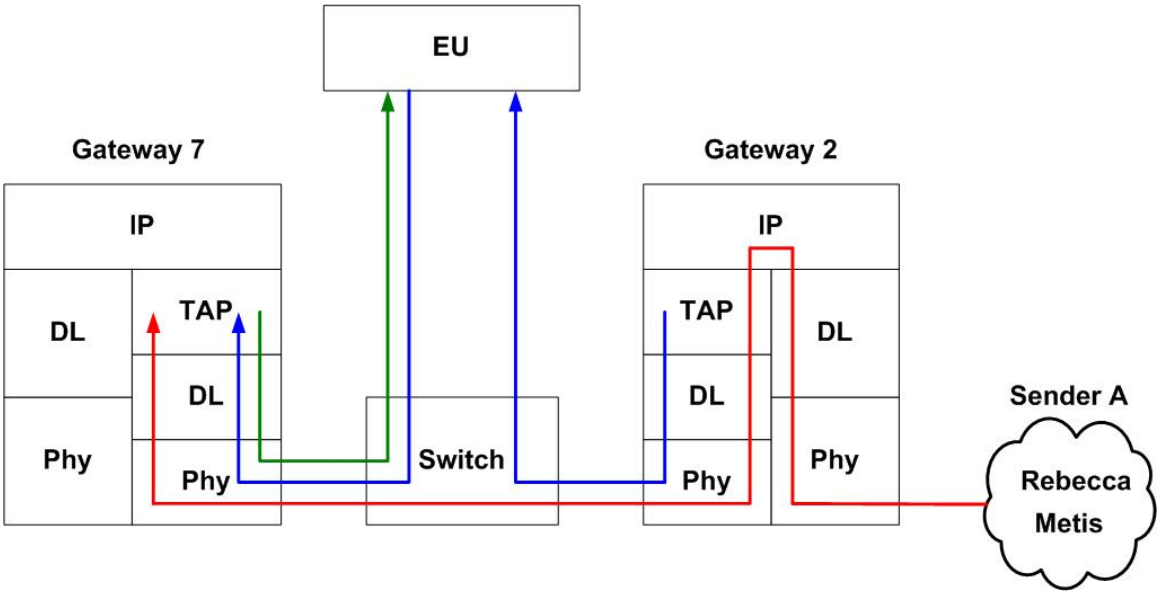


Figure 10 : Overview of ACE Gateway Protocol Stack

Tcpdump

Tcpdump is a powerful tool that allows us to sniff network packets to make statistically analyzes of the traffic. It monitors all traffic, with the according timestamps. Users can decide if the packet headers should be printed to the screen on the fly, or if packet headers and packet data should be saved to a file for later and more detailed analyzes. There are several flags to set in tcpdump, which gives opportunity to make detailed expressions to dump exactly the desired traffic. Tcpdump works in Linux, the Windows version is called Windump. Tcpdump is available at http://www.tcpdump.org/.

Tcptrace

Tcptrace is a tool for analyzing TCP dump files, like Tcpdump and Windump. It can produce several different types of output containing information on each connection seen, such as elapsed time, bytes and segments sent and received, sacks, retransmissions, round trip times, window advertisements, throughput, and more. It can also produce a number of graphs for further analysis. Output for our analyses are 'unique bytes sent', 'data transmit time' and 'throughput' respectively, and files for graphs. Tcptrace is a freeware and can be downloaded from http://jarok.cs.ohiou.edu/ software/tcptrace/useful.html.

Xplot

Xplot is a very useful and simple software for displaying graph files in Linux. It is typically used together with Tcptrace, to display the xpl-graphs. Despite its simplicity, it displays a very detailed picture of the graphs. Each segment, acknowledgement, sack, retransmitted segments are all given different colours so we easily can get an overview of the transmission. Xplot is available at http://www.xplot.org/.

Iperf

Iperf is used to measure the TCP and UDP bandwidth performance over a link. One computer is set as server, and another as client. Several parameters can be set to define the link, like buffer size, window size, segments size, total data to transmit or total time to transmit and UDP. During tests, Iperf can generate the set amount of data, so users do not have to pre-generate a file to send. Ipfer reports bandwidth, delay jitter and datagram loss. Iperf available at http://dast.nlanr.net/Projects/Iperf/

Fedora Core 4

Fedora is the new Red Hat Linux, and it is developed and distributed by Red Hat Inc. It is based on Linux, and all software included is open source and free to use. The distribution is released twice a year, and is therefore most suitable for desktop computers. It is very easy to update and install new software, because Fedora uses the Red Hat Packet Manager (RPM). Users can install software directly from the Internet with these packets. Fedora runs on different Linux kernels, even though the kernels operate differently. Fedora can be downloaded from http://www.redhat.com/fedora/

Linux Kernels

The advertised windows are set differently in the different kernel versions. Kernel version 2.2 set the advertised windows to the half of the size as the TCP buffers. In the contrary kernel version 2.4 and above, do not. In kernel 2.4, there is implemented a new dynamic autotuning mechanism. This mechanism estimates the bandwidth every RTT, to derive the receivers' window. Autotuning is controlled by the new kernel variables 'net.ipv4.tcp_rmem/wmem'. The senders' window size is not constrained, but will grow until throttled by the receivers advertised window. Linux 2.4 moderates CWND growth by increasing CWND only if the INFLIGHT data is greater than or equal to the current CWND. When setting the buffer sizes to 16000 bytes, we have to take into account that Rebecca runs on kernel version 2.4, and set the windows to 8000 bytes instead.

5 Emulation of Ad Hoc Networks

Our results will relay heavily on the models used in the emulation of the ad hoc network. It is of great importance that the models are similar to the genuine network, and that our constraints are reasonable. The ACE system can emulate a network in two different ways, using two different methods. The first model is the two state Markov model. This model uses two states to describe the link: a bad state (no transmission) and a good state (perfect transmission). In the second method, which we have called a characteristic model, the probability of packet loss P_{pd} is calculated on the basis of IEEE 802.15.4 characteristics and a mobility model. Each packet has a probability equal to P_{pd} to be dropped.

The two state Markov model has been used in numerous prior works, and is a good model for link simulation. However, we decided to use the characteristic model, even though this was a more complex and time consuming task. The reason for this choice was to have an ad hoc emulator for future work, also, with this model it is easier to modify the IEEE 802.15.4 characteristics. To implement this model on the ACE system we had to change the C/C++ code. The existing code is performance optimized and very dynamic, thus, also very complex. Because of our lack of experience in C and C++ programming, this task was harder and more time consuming than first anticipated. When the modifications were complete, we experienced problems compiling the files on the emulator. The files compiled perfectly separated, but not together. The problem was caused by interconnection of C and C++ files. To fix this problem would take some time, after the problems had been solved, we would also have to do an application test to verify that our modifications gave the correct and reasonable test results. As the work drew closer to the end, we evaluated the situation, and decided to change to the two state Markov model. The change gave us more time to thoroughly test and evaluate the TCP protocols, which is the primary task of this work.

The two state Markov model and the characteristic model are described in sections 5.1 and 5.2, respectively.

5.1 Two State Markov Model

The two state Markov model uses a Markov chain to describe the link. The Markov chain is a discrete-time stochastic process. These stochastic processes are memory less, so at each time step, the model knows witch state it is in, but not in which state it previously has been. Each state has a certain probability to change to another state. The change from a state to another are called transitions, transitions are based upon current, not past, information.

As mentioned, uses our Markov chain two states to describe the link: a bad state (no transmission) and a good state (perfect transmission). To give an example, let's say that the good state has a 90 % chance to stay in the good state, thus 10 % chance to change to bad state. The bad state has a probability of 50 % to stay in the bad state, and 50 % chance to change to the good state. If we computed this chain we might get a result like this: good – good – good – good – good – good – bad – bad – good – good – good – good – bad – good. By using such a model to describe the link, errors

are randomly injected into the channel; consequently the duration of good and bad state varies randomly. The real transition probabilities used in our testing are presented in chapter 6 along with other scenario aspects.

To emulate this model we connected a TCP sender and a TCP receiver to the ACE system. In emulation the TCP sender sends a segment to Gateway 1, which forward it to Gateway 2, as indicated with arrow 1 in Figure 11. It immediately sends a Control Packet (CP) to the EU, indicated as arrow 2, which contains the address of Gateway 2. The EU decides if the packet is dropped or not according to which state the link is in. The outcome of this decision is sent to Gateway 2 in a CP. If the link was in a good state Gateway 2 forwards the segment to the TCP receiver, if the link was in a bad state the segment is dropped.

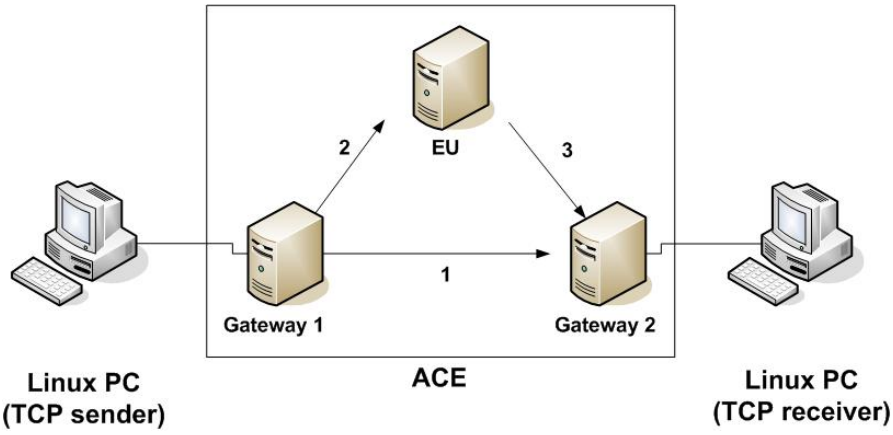


Figure 11 : Emulation of the two state Markov model

5.2 Characteristic Model

To implement the Characteristic model on the ACE system the C and C++ code had to be changed. The Characteristic model had two major differences compared to the exciting model: the calculation of the P_{pd}, and the implementation of a mobility model. To modify the calculation of P_{pd} was simply to change and add some parameters, while the implementation of a mobility model required a more extensive change. Before the EU decide if the packet is dropped or not it has to know the position of both sender and receiver. We wanted the GTWs to calculate their movements and positions, thus both parts must inform the EU of their position. If we look at Figure12 and consider the left Linux PC as the TCP sender, the TCP segment will be sent from the Linux PC to Gateway 1. Gateway 1 forwards the segment to Gateway 2, as indicated in Figure 12 as arrow 1. Then Gateway 1 sends a CP to the EU to inform the EU that a segment has been sent, this is indicated as arrow 2. The CP contains information about the segment, addresses to the communicating gateways and the location of the sending station. When Gateway 2 receives a segment, it immediately sends a CP to EU, as indicated as arrow 3. This CP is similar to the CP sent by Gateway 1, except that it contains the location of the Gateway 2. When EU has received a CP from both sender and receiver, the distance between the two stations is calculated. Based upon this distance and the P_{pd} the EU decide if the segment should be dropped or not, and if not, at which time it is received. When the decisions are made, a CP is sent from the EU to the Gateway 2 with information about the outcome of this decisions, this is indicated as arrow 4. If the EU decides that the

segment should be received, Gateway 2 forwards the segment to the TCP receiver, if the segment should be dropped, Gateway 2 does nothing.

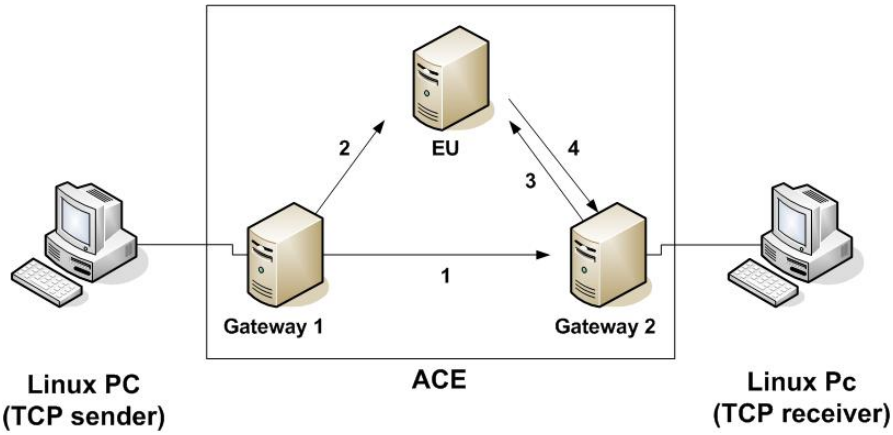


Figure 12 : Emulation of the Characteristic Model

We have seen that the Characteristic model requires that the receiving gateway sends a cp to the EU with position information. On the original ACE implementation this is not necessary, because every communication part is stationary. We could avoid this by letting the EU calculate the movement of every node. This would simplify our implementation, but has some back draws. It would require more processing for the EU, for now it would not make much difference, since we only use a few gateways. But in the future I might be desirable to add a large number of virtual devices. This would also make the emulation unit less dynamic, and the emulation would have a stronger simulation resemblance.

Our mobility model is presented in the next section. In section5.2.2 the calculation of P_{pd} is described. The implementation of the Characteristic model could be a thesis project itself, while we are more concerned about TCP performance over error prone networks. Thus, quite a few constraints have been made in this emulation. The last section gives an overview of the constraints which had to be made.

5.2.1 Mobility

For wireless ad hoc networks, the mobility of the nodes is one of the most important factors for determining the channel performance. There are two ways to describe movement of nodes, either by real life traces, or simulation. Using real life traces of genuine up to date system are the most accurate, but it requires both resources and time. Simulation can easily be done by creating a simple code. Many algorithms have been proposed for simulation of node mobility, and they can be categorized into two groups, group mobility models and entity mobility models [30]. For group mobility models, each node is a member of a group, and it moves dependently of the other nodes. A real life example is a group of soldiers moving through an area. In entity mobility models, a node moves in dependent on the other nodes. A real life example of these models is people walking around in a market.

For our work we decided to use the entity mobility model. We chose this model because we want to describe the effect of movement for each node independently of

the others, and to see how movement affects the implementation of different transport protocols.

Entity Mobility Models

Many algorithms are suggested for entity mobility models. Some of these models are described below[30, 31];

Random Walk Mobility Model: This model is also known as the Brownian Motion Mobility Model. This model comes in many forms and derivatives. The basic idea is that each node randomly moves around, in a specified area, with random speed. The speed is typically a random number between a given minspeed and maxspeed, while the angle that describes the movement is a random angle between 0 and 2π. Both values are uniformly distributed. For each movement, the model also defines the number of jumps or time units the node will walk in the current direction. If a node tries to cross the boundaries of the area, it will be given a new direction of movement so it does not cross these boundaries. The Random Walk Mobility Model is either time based or step based. However, the model does not store information about prior speeds and angles; this might produce unrealistic changes of speed and direction [30]. This problem is solved in other entity mobility models, like the Gauss-Markov Mobility Model.

Random Waypoint Mobility Model: This model is quite similar to the Random Walk Mobility Model. The main difference between the two is that the Random Waypoint Mobility Model also includes stop time between each change in direction. The stop time is a random pause. The relationship between pause time and speed highly affects the stability of the network.

Random Direction Mobility Model: When using the Random Walk Mobility Model and the Random Waypoint Mobility Model, in some situations the nodes tend to cluster in the center of the simulation area. The Random Direction Mobility Model tries to cope whit this problem. Nodes in this model are forced to walk all the way to the boundaries of the simulation area before they can change direction. This model is similar to the Random Waypoint Mobility Model in all other aspects.

Gauss-Markov Mobility Model: This model is somewhat similar to the Random Walk Mobility Model. The purpose of this model is to avoid sudden changes of direction and sharp turns. The Gauss-Markov Mobility Model uses a parameter α, where α is a number between 0 and 1, to determine the degree of randomness of the nods movement. 0 is totally random and the movement will be identical to the Random Walk Mobility Model, while 1 is a fixed number and the node will walk in a straight line. The equation used for calculating the movement takes prior movement in to account, thus creates a more realistic movement.

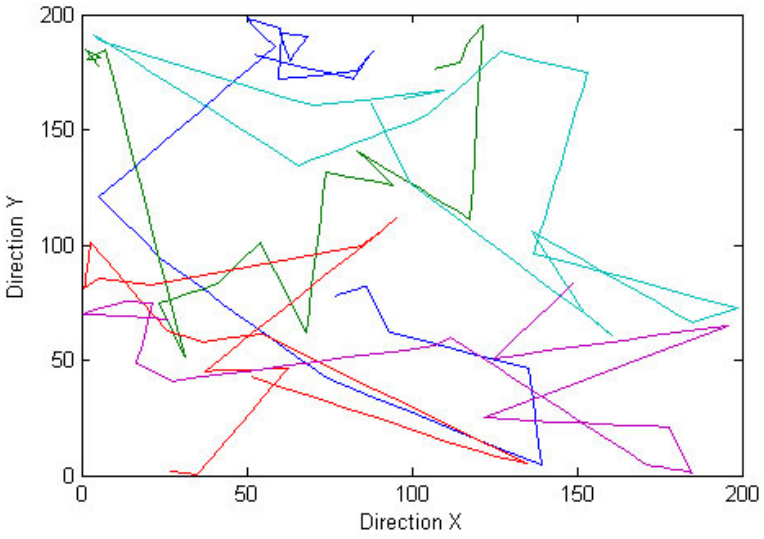
Random Trip Mobility Model: The purpose of this model is to create a more realistic movement of nodes within a city environment. This model is defined by a set of paths over a connected domain, an initiation rule and a trip selection rule [31]. The set of paths are defined as a number of waypoints connected together, where the paths are the “roads” from one waypoint to another. A node gets an initial position, set by the initiation rule. After an initial position is set, the trip selection rule chooses a random trip, consisting of one or more paths, which are traversed whit a random speed.

Our Mobility Model

We decided to implement the Random Waypoint Mobility Model. Due to the problems described in the previous section we made some modifications. This model is relatively easy to implement, it is also used in numerous simulations and is widely recognized. Our model is step-based and not time-based, so real time is not implemented in our model. We use time units to represent time; for each time unit, all nodes either move or pause. The initial positions of the nodes are uniformly distributed around the whole simulation area. After initialization, all the nodes immediately start to move. We use three random variables to generate the movement; velocity, angle (0-2π) and a variable we call time_in_vector. The velocity is the speed of the nodes and angle generates directions. The last presented, time_in_vector, describes for how many time units a node is in a vector. When a node starts to move, it randomly chooses velocity, angle and time_in_vector with a uniform distribution. Velocity and angle are constant until a waypoint is reached. Time_in_vector is a counter, which is decreased for each step. When time_in_vector is 0, the node has arrived at its waypoint, and pauses for a random time. When the pause time is over, the three random variables are generated again, and the node starts to move. The random generator is uses microseconds as seed, which it gets from the system clock by using the datetime function.

Before a node takes a step, it checks if the step will exceed the boundaries of the simulation area. If the boundaries will be exceeded, the node stops and pauses, instead of moving. When the pause time expires, it finds new movement parameters. These parameters are checked, and it is made sure that the new waypoint, not only the next step, is inside the simulation area. Notice that this mechanism is only used for the first waypoint after the node tries to exceed the boundaries. The purpose of this mechanism is to avoid too many stops at the edge of the simulation area.

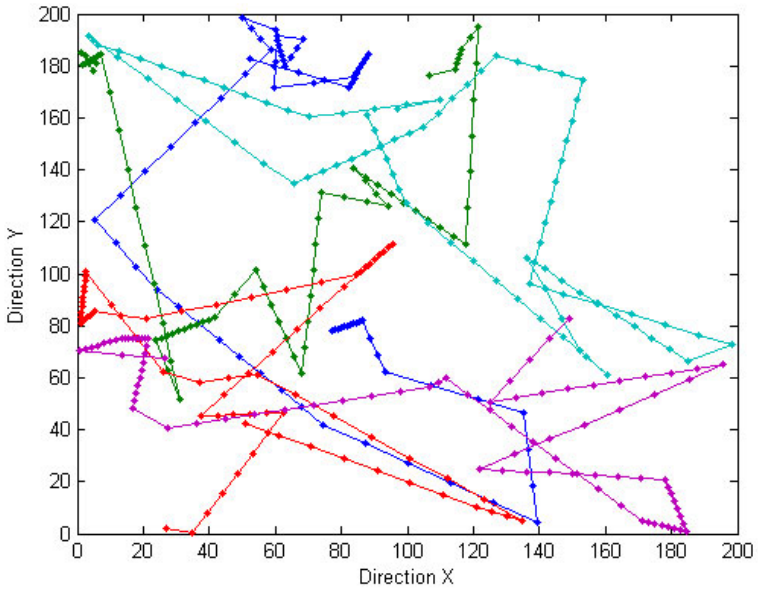
Our mobility model has no limits with regards to the number of nodes or the number of steps.



Graph 2 : Node Mobility

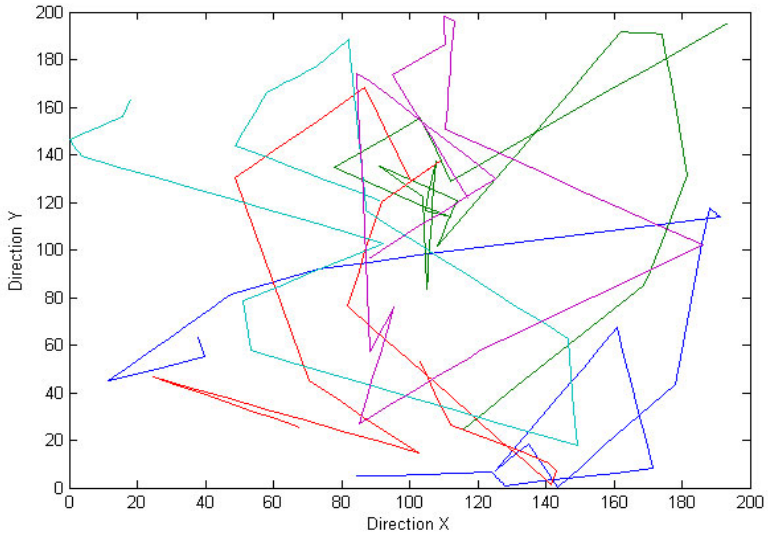
Graph 2 shows the movement of 5 nodes over 200 steps. The simulation area is limited to 200 in both x and y direction. Graph 3 shows the same movement, but

plotted with a dot for each step. For each vector (straight line) the length of the steps (distance between dots) may vary. This length describes how far the node moves within one time unit, where one time unit could for instance represent one second or one millisecond. If the length of a step is large, the node is moving fast (high velocity), if the length is short, the node is moving slowly (low velocity). For each vector the speed is the same. The pause time can however not be seen in this graph, as this is represented as dots in exactly the same positions.

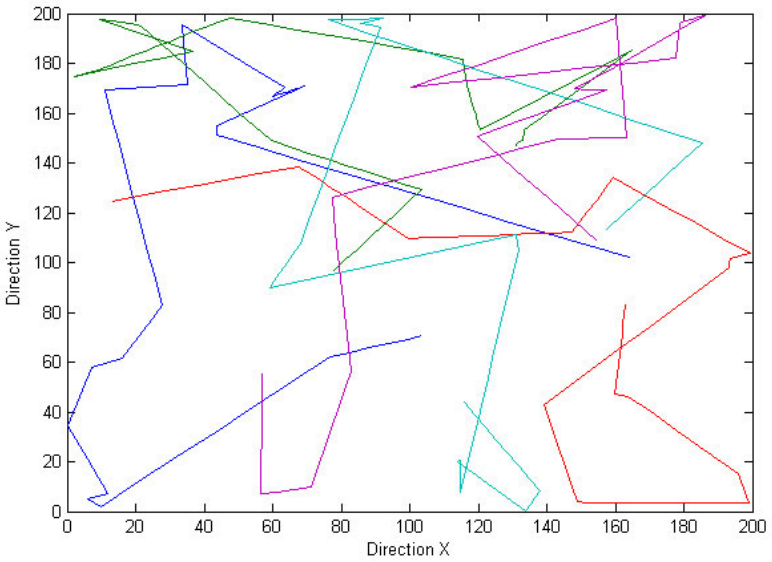


Graph 3 : Node Mobility with Steps

As mentioned previously, the angle of movement of a node in the random waypoint mobility model might sometimes be very small. The small angles generate unnatural movement. To cope with this problem we generate the angles using a normal (Gaussian) distribution. The initial angles are uniform distributed. The following angles are normally distributed around the last angle, with a varying standard deviation. In this way the node will take less sharp turns. In Graph 2 and Graph 3 the angles are uniform distributed. In Graph 4 the node is normally distributed with the mean equal to the last angle, and the standard derivation equal to 3. In this graph the number of small angles are reduced compared to Graph 2 and Graph 3. In Graph 5 the standard deviation has been decreased to 1.5, this results in yet more reduction of small angles. The angle which is generated after the node hits the simulation area edge also uses the last angle as mean. If the angle will move the node outside the simulation area, the angle is disregarded, and a new random angle is generated. If the standard deviation is low, it might take some time before an acceptable angle is found. For instance, let's say a node has the initial position in the centre of the simulation area. The first angle it gets is 0, and it moves in this direction until it reaches the boundaries of the simulation area, and hits the edge with an angle of 90 degrees. It pauses and then tries to find a new angle normally distributed around 0. The angles from 0 to $\pi/2$ and $3/2\ \pi$ to 0 are not allowed. If the standard deviation is sat to 0,5, only a few percentage of the generated angles are in the acceptable area. This problem could be solved by forcing the angle to be randomly generated of a range of only acceptable angles.

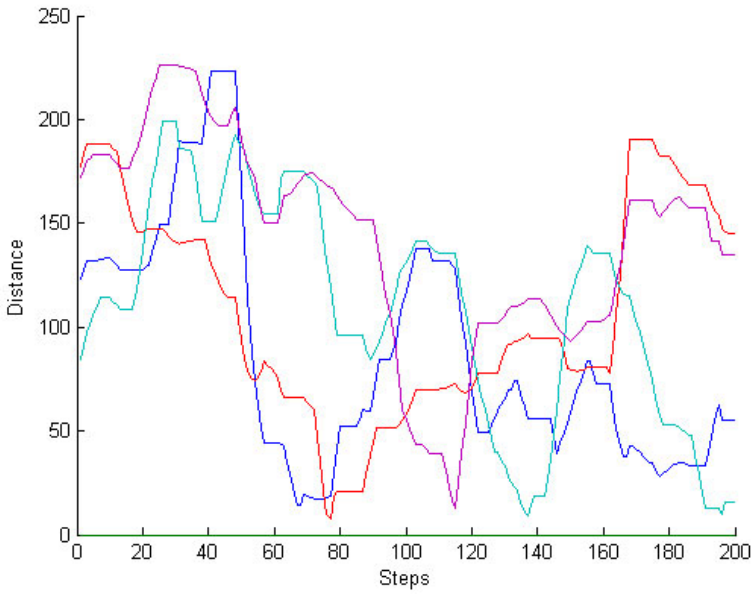


Graph 4 : Node movement with μ = 3



Graph 5 : Node Movement with μ = 1.5

Graph 6 shows the distances from one node (the node represented with green colour in Graph 2 and Graph 3) to the four other nodes. It should be noted that the node which the distance is calculated from is also moving. Thus, this graph does not represent movement from one stationed sink to surrounding mobile nodes.



Graph 6 : Node Distances

5.2.2 Channel Modelling

Packet drop is the most important parameter for our TCP performance evaluation. This is due to TCP Congestion Control mechanism which is described in chapter 3. This section gives an overview of some aspects of our channel model, with focus on packet drop. First some parameters are presented, and then the calculation of the probability of packet drops is described with mathematical equations, which have been implemented in the emulator. The last section is a description of how the probability of packet loss is evaluated.

Link Parameters

The link parameters have been set according to the IEEE 802.15.4 specification.
We decided to use the 2.4 GHz frequency band. This band is one of the so called Industrial, Scientific and Medical (ISM) frequency bands. Below is a table which indicates some of these parameters [22].

PHY (MHz)	Frequency band (MHz)	Spreading parameters		Data parameters		
		Chip rate (kchip/s)	Modulation	Bit rate (kb/s)	Symbol rate (ksymbol/s)	Symbols
868/915	868–868.6	300	BPSK	20	20	Binary
	902–928	600	BPSK	40	40	Binary
2450	2400–2483.5	2000	O-QPSK	250	62.5	16-ary Orthogonal

Table 2 : IEEE 802.15.4 Parameters

Table 2 shows other parameters used. Each parameter is described together with its use in section 5.2.2.

Other Parameters		
Symbol	**Description**	**Value**
R	Coderate	1
Tx Power	Transmission Power	10 mW
G	Antenna gain	1 mW
$\Phi(\omega)$	Power Spectral Density (PSD)	Bandwidth/2
k	Boltzman constant	1.380 6505×10^23 joules/K
T	Noise Temperature	190 K
n	Packet Size	60 kB

Table 3 : Other Ad Hoc Network Parameters

Calculation of Probability of Packet Drop

In this calculation many simplifications and approximations have been used. This has been done to avoid long computation time.

For the 2.4 GHz band, O-QPSK modulation is proposed in the specification. We use Rayleigh fading to describe the propagation in the environment. Rayleigh fading is a reasonable model when there are many objects in the environment that scatter the signal before it arrives at the receiver.

Before we can calculate the probability of packet drop (P_{pd}), we have to calculate the probability of bit error (P_{be}). For simplifications we used the formula for BPSK modulation; this modulation is also based on the PSK principles and is very similar to O-QPSK modulation. For BPSK modulation over a Rayleigh fading channel the P_{be} is equal to [32]:

$$P_{be} = \frac{1}{2}\left(1-\sqrt{\frac{\gamma R}{1+R\gamma}}\right) \quad (1)$$

Here R is equal to the code rate. No error correction scheme is specified for IEEE 802.15.4, thus R has been neglected because it is equal to 1. γ is the signal to noise ratio (SNR), and given by:

$$\gamma = \frac{E_b}{N_0} \quad (2)$$

The SNR is a term which describes the quality of a signal. This is the ratio between the wanted signal energy per bit (E_b) and the disturbance created by the surrounding environment, so called noise (N_0). E_b is the bit energy and is given by:

$$E_b = Tx\,Power \times \frac{1}{Bandwidth} \times G \times L \quad (3)$$

In equation 3 Tx Power is the transmission power. A genuine device is expected to operate with transmission power of -3 to 10 dBm, with 0 dBm being typical [22]. As

the battery power in a device decreases, so will the transmission power. For simplifications we have used a constant, maximum transmission power which equals 10 dBm (10 mW). Tx Power is multiplied by the bit duration. The bandwidth is set to 250 kb/s, according to Table 2. G represents the antenna gain. We assume that none of the devices have antenna connectors, and therefore can be interpreted as effective isotopic radiated power (EIRP). The devices have then 0dBm antenna gain, which is equal to 1 mW. The last element in equation 3 is the attenuation L and is given by:

$$L = \frac{4 \pi d^2}{\lambda^2} \tag{4}$$

In Equation 4 d is the distance between two nodes, of which one is sending and one receiving. This distance is calculated according to the mobility model described above. λ refers to the wavelength. λ is given by the speed of light (c) divided by the frequency (f). The speed of light is approximately 3×10^8 m/s, while the frequency is 2.4 GHz (2.4×10^9 Hz). Now the calculation of the bit energy is done, and we return to equation 2. To complete the calculation of the SNR we need to find the noise level N_0, which is given by:

$$N_0 = \Phi(\omega) \times k \times T \tag{5}$$

In this equation $\Phi(\omega)$ is the power spectral density (PSD). The PSD describes how the power of a time series is distributed with frequency. This is usually derived from the Fourier transform. PSD of a signal is the square of the magnitude of the Fourier transform of the signal. In our case the PSD can be approximated to the half of the bandwidth, this approximation is very simplified, but is a good approximation. k refers to Boltzmann's constant, which is equal to $1.380\ 6505 \times 10^{-23}$ joules/Kelvin. The last variable in this equation is T. T refers to the noise temperature, which is the temperature of a resistor that has noise power equal to that of the device or circuit. This phenomenon is summed up as noise temperature. The exact value for T can be calculated through advanced functions and equations, but usually a static number is used. For simulating these types of devices it is common to use a noise temperature of 190^o Kelvin, which we also decided to use. Now that we have found the noise level, we can derive SNR using equation 2, and then calculate the probability of bit errors.

Our goal for this calculation is to find the probability of packet drop, P_{pd}. The codeword error probability can be evaluated as follows [32]:

$$P_{pd} = \sum_{i=t+1}^{n} \binom{n}{i} p^i (1-p)^{n-i} \tag{6}$$

Here p represents the bit error probability derived from equation 1. n is the number of bits in the packets, in other words the packet size. t is the number of correctable bits. As mentioned earlier no error correction coding is used, so t equals 0. To calculate this equation we have to calculate the faculty of n. Because n is such a large number, this calculation requires a lot of computation time and resources.

Since i start at 1, equation 7 yields:

$$P_{pd} = 1 - (1 - p)^n \qquad (7)$$

As in equation 6 p is the probability of bit error derived from equation 1 and n is the packet size. By computing the equation 7 we will find an approximation for the probability of packet drop.

Evaluation of Packet Drop

To evaluate if a packet is dropped or not, a two-state model is used. In the first state the packet is received without problems, in the second state the packet is dropped. The emulator first calculates the probability of packet drop (P_{pd}) as described above. P_{pd} is used as a threshold to separate the two states. The emulator will generate a random uniformly distributed number (rand) between 0 and 1. If rand is larger than the probability of packet drop (rand > P_{pd}) the packet is received without problems, i.e. the first state. If the random number is less then the probability of packet drop (rand < P_{pd}) the packet is dropped, i.e. the second state.

5.2.3 Constraints

These constraints have been enforced when creating the mobility model:

Access: no access scheme is implemented. An access scheme should generally be implemented in emulations and simulations, which seek to imitate genuine radio or wireless communication. These schemes have a major effect on communications where many devices try to connect to one network device, e.g. many ad hoc devices that try to connect to one sink. If two devices try to use the link at the same time, the signals will be mixed and unreadable for the receiving network device. If collision occurs the devices must retransmit the packets. In addition access mechanisms require extra signaling. The physical structure of the emulator limits the number of nodes to 8. Thus, collisions would not occur so often. In sensor network the nodes transfer a relative small amount of information, thus the packet size is small. Each node requires only a small portion of the bandwidth during transmission, which also will reduce the probability of packet collisions. Due to these facts, the access scheme would not have a major effect on our results.

In 802.15.4 specification use the Carrier Sense Multiple Access with Collision Avoidance as access scheme, while the TDMA scheme is implemented in the ACE system.

Routing: routing in an ad hoc network is a complex task. Our ad hoc network model is limited by nodes and we have limited the number of paths a packet can take through the network to one. Thus, nodes do not need to store and evaluate routing information. For this reason we have neglected routing.

Calculation of packet drop: as mentioned in section 5.2.2 many constraints have been taken to simplify the calculation of packet drop. Beside the calculation itself, the

constraints include transmission power, noise temperature and power spectral density. A proper model for battery lifetime and energy levels could also be implemented, but we concluded that such a model would have minor effects on the performance evaluation. Battery lifetime and energy consumption affect the transmission power. In this work, however, for the sake of simplicity, the impact of energy consumption on the performance has been neglected, and therefore the transmission power has been assumed constant.

6 Measurement Concept

The TCP versions were tested over a dedicated links using just one connection, with no other traffic present. In such scenarios it is relatively easy to evaluate the performance and the behaviour, because there are less variables. Using an open link, with traffic shared with other users would create a more realistic scenario, but on the other hand, the results would vary a lot, which would cause the results to be spread and hard to evaluate. To oppose this we would had to increase the number of trials to several hundreds, or even thousands of trials. Another solution would of course be to start multiple defined connections over the dedicated link. This would be an interesting task where we could measure friendliness and performance over a shared link including a variety of protocols. But because of our time constraints we will leave this up to others.

Our evaluation of performance is based on the throughput. The throughput describes how much data is transferred over the link, usually in bytes pr seconds or as an average. Setup and teardown (Syn and Fin packets) of the TCP connection have not been included in the calculation of the throughput. We only look at the bulk throughput performance witch is dominated by the data transfer phase.

The next two sections describe individually the scenario for the satellite network and the ad hoc sensor network.

6.1 Satellite

In the satellite test, the TCP sender was in Naples and the receiver was in Genoa. The receiver in Genoa registered the data by using the Tcpdump, so the registration of TCP traffic is based on segments received by the receiver and not the ACKs received by the sender.

The tests were run on a dedicated 2 Mb/s link. Packet headers, coding schemes and modulation add redundant data to the transmission. If we disregard this redundant data we will get the effective bandwidth, which is measured to approximately 1.2Mb/s. The packet size has been sat to 1500 bytes, which is a standard size for IP packets. By subtracting the IP and TCP header we have a TCP payload of 1448 bytes (1550 bytes – 20 bytes – 32 bytes).

In the trials 10 MB (10486416 bytes) of data were sent, so each trial has a variable duration. 10 MB was chosen simply because it gave reasonable trial durations, which varied from 100 seconds to 270 seconds, with and average of approximately 150 seconds.

A total of four TCP versions were tested: TCP Hybla, TCP NewReno with SACK, TCP Vegas and TCP Westwood+. TCP NewReno is always interesting to take into account, since it is the most common TCP implementation. TCP Hybla, TCP Vegas and TCP Westwood+ are all TCP version which are expected to perform well over a satellite link.

Table 4 shows a summary of the parameters in the measurement concept.

Scenario Parameters for the Satellite Link	
TCP Veriants:	TCP Hybla, TCP NewReno with SACK, TCP Vegas, TCP Westwood+
Registered at:	Sender
Packet Size:	1500 bytes
TCP Payload Size:	1448 bytes
Amount of data in each trial:	10 MB
Period of each trail:	Variable

Table 4 : Summary of Satellite Measurement Concept

6.2 Ad Hoc Sensor Network

This section describes the scenario for the ad hoc sensor network. In the network we emulate, data is sent from wireless sensors, through a defined channel, to a sink which again forwards the packets to the satellite link. In this scenario, we just have one sender and one receiver at the same time in an enclosed environment. This way the sender does not have to share the bandwidth, and can send with the highest throughput possible.

For this scenario, we decided to test two different TCP versions; TCP NewReno and TCP Westwood+. Both versions are well-known, and are used in different networks today. They are both including Congestion Control and the SACK-option, but are still slightly different in the way the work. A more detailed description of TCP NewReno and TCP Westwood+ is given in section 3.1.3 and 3.1.4, respectively.

We sat the bandwidth to 250 kbps. This bandwidth is standard for the IEEE 802.15.4 Physical Layer, when using the 2.4 GHz frequency. The 2.4 GHz band operates worldwide, and is often used for wireless communication [20]. Wireless sensors transmit small amount of data relatively often, with the shortest transmission time possible. Maximum segment size for IEEE 802.15.4 is 128 bytes [20]. We decided to set the maximum segment size, MSS, to 60 bytes. This results in a payload size of 40 bytes, since the TCP header is 20 bytes. Additional, the sensors have limited memory, so we decided to set the TCP buffer sizes, for both receiving and sender buffer, to 16000 bytes. Thus the advertised window size will be 8000 bytes. Since we emulate communication between two similar devices, is it important to have the same configurations on both TCP sender and TCP receiver. The advertised window size is in Linux kernel 2.2 automatically set to the half of the TCP receiving buffer size, but we had to manually set the advertised window size to 8000 bytes in kernel 2.4. Before testing, we had to set the delay for the channel. We had to take into account different quantity of nodes, different sizes of test area and piconets. We wanted to emulate a multihop ad hoc network. Woon and Wan have in [14] tested multihop transmission with bandwidth 250kbps, packet size 60bytes and a transmission range at 15m. We decided to use a delay of 10 ms, 30 ms and 50 ms, according to Woon and Wans results, this corresponds to networks with 2, 5 and 8 hop routing, which is reasonable sizes of genuine sensor networks.

We used Iperf to generate the traffic, and to measure the bandwidth. Iperf default sends data and runs test for 12 seconds, to calculate the bandwidth. But in our case we wanted to have equal tests, with different delays and packet loss, independent of time. We decided to test with an equal amount of sent data instead of measuring bandwidth in a set period. With the relatively low bandwidth and the small packet sizes, we decided to set the file size to 1 Mbyte. We estimated the test to last from 150 to 1200 seconds, depending on the different parameters, which is a reasonable time to get accurate results.

We used the two state Markov model to simulate the channel. The model use two parameters, p_{GB} and p_{BG}, as transition probabilities between the good state (transmission) and the bad state (no transmission). The transition matrix for our model is illustrated in (1).

$$M_c = \begin{pmatrix} p_{BB} & p_{BG} \\ p_{GB} & p_{GG} \end{pmatrix} \quad (1)$$

In (1) p_{BG} is the probability of transistion from bad (B) to good (G), and p_{GB} is the probability of transition from good to bad. While p_{GG} and p_{BB} are the probability that the model will stay in its current states: probability of staying in good when in good and staying in bad when in bad, respectively. Figure 13 also illustrates the transitions between the states.

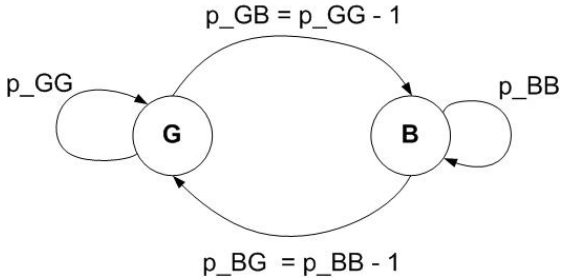


Figure 13 : Transitions in the Two State Markov Model

We used the transition probabilities from [13], which refer to [33] for the calculation of p_{GG} and p_{BB} using a simplified analytical model for calculation. Table 5 gives an overview of the p_{GG} and p_{BB}, and the corresponding values:

$f_d T$	P_E	$F(dB)$	p_{GG}	p_{BB}	$1-p_{GG}$ (α)	$1-p_{BB}$ (β)	$(1-p_{BB})^{-1}$
0.01	0.001	29.998	0.99933	0.32945	0.000671	0.670548	1.49132
0.01	0.01	19.978	0.99752	0.75431	0.002482	0.245692	4.07013
0.01	0.1	9.7732	0.99187	0.92685	0.008128	0.073149	13.6708
0.08	0.001	29.998	0.99901	0.00824	0.000993	0.991761	1.00831
0.08	0.01	19.978	0.99069	0.07729	0.009314	0.922714	1.08376
0.08	0.1	9.7732	0.94035	0.46319	0.059646	0.536812	1.86285
0.64	0.001	29.998	0.999	0.00119	0.001	0.998815	1.00238
0.64	0.01	19.978	0.94035	0.01183	0.059646	0.988166	1.01198
0.64	0.1	9.7732	0.90182	0.11638	0.09818	0.883624	1.1317

Table 5 : Transition Probabilities for Two State Markov Model

The values for p_{GG} and P_{BB} are calculated on the basis of the normalized Doppler bandwidth (f_dT) and the fading margin (F) represented in dB, these values are the physical representation of the link. The normalized Doppler bandwidth is given by the equation:

$$f_d T = f_0 \times \frac{v_0}{c_0} \times T \qquad (2)$$

Here f_0 is the frequency, which is 2.4 GHz, v_0 is the velocity of the node, c_0 is the speed of light which is approximated to 3 * 10^8 m/s, and the T is packet duration. In our case, the transmission rate is 250 kbps and the packet size is 60 bytes. Packet duration is (8 * 60 bytes) / 250000 bps = 0.00192 s, or approximately 2 ms. By evaluating (2) we can derive an expression of the node velocity v_0. For the three values for f_dT: 0.01, 0.08 and 0.64, the corresponding velocities are 0.65 m/s, 5.21 m/s and 41.67 ms, respectively. The fading margin is a parameter which describes the total fading of the system, and includes antenna heights, gains, link loss, transmit power and receiver sensitivity. The average packet loss (p_E) is given for each of the corresponding values of p_{BG} and p_{GB}. p_E is dependent both on the normalized bandwidth and fading margin.

Table 6 gives a summary of the parameters used in the ad hoc sensor network scenario:

Scenario Parameters in the Ad-hoc Sensor Network	
TCP versions:	NewReno, Westwood+
Bandwidth:	250 kbps
Maximum Segment Size:	60 bytes
TCP buffers:	16000 bytes
Advertised Window:	8000 bytes
Filesize:	1 Mbytes
Delay:	10ms, 30ms and 50ms

Table 6 : Summary of Ad Hoc Sensor Network Measurement Concept

7 Results and Discussion

This chapter is devoted to the performance analyses. The description is two-folded: firstly the satellite link has been investigated and after the wireless ad hoc link. It is important to clarify that in all the tests, on both the satellite and the emulator, the receiver side registered the data. The TCP sender and the TCP receiver will have their own comprehension of what happened during the transmission. This is due to the lack of knowledge to why packets are dropped. If the data is registered at the sender side, the CWND will show a higher value than actually experienced, due to the fact that the sender is not aware of the actual success of the transmission until an ACK segment is received. On the contrary, if the transmission information is registered at the receiver side, a more accurate throughput value will be registered, because the sender side will take in to account RTT, propagation delay and bottlenecks. The drawback of registering at the receiver side is that the number of retransmissions may not accurately reflect the number of lost packets. In this case, the number of retransmission is the number of packets which are received at least once, not lost. The number of out of order packets indicates lost packets.

7.1 Satellite Link

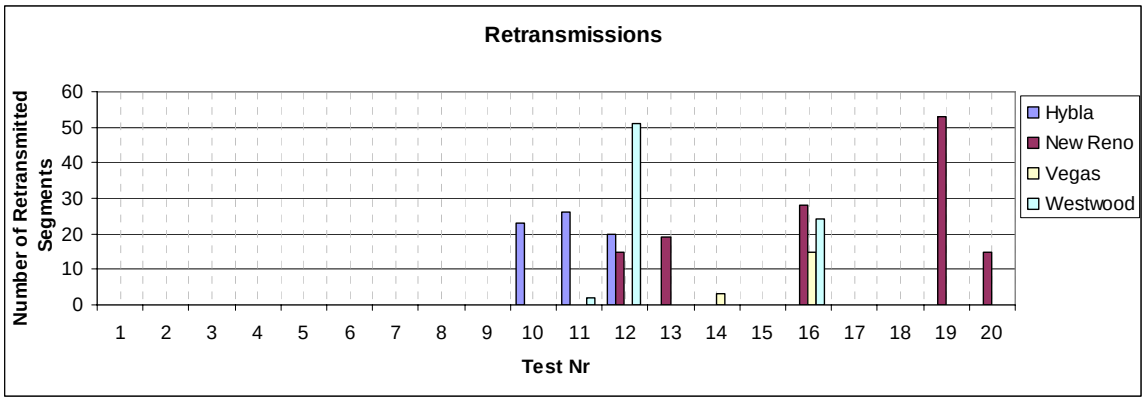
20 trials were run for each of the four TCP variants, divided over two different days. The first day trials 1 to 10 were run for each TCP variant, and then a few days later trials 11-20 were run.

7.1.1 Link Degradation

We experienced different results for the two days, caused by the different degradation in the link. The degradation is usually only caused by packet drops and retransmissions. In our testing we also experienced many out of order segments.

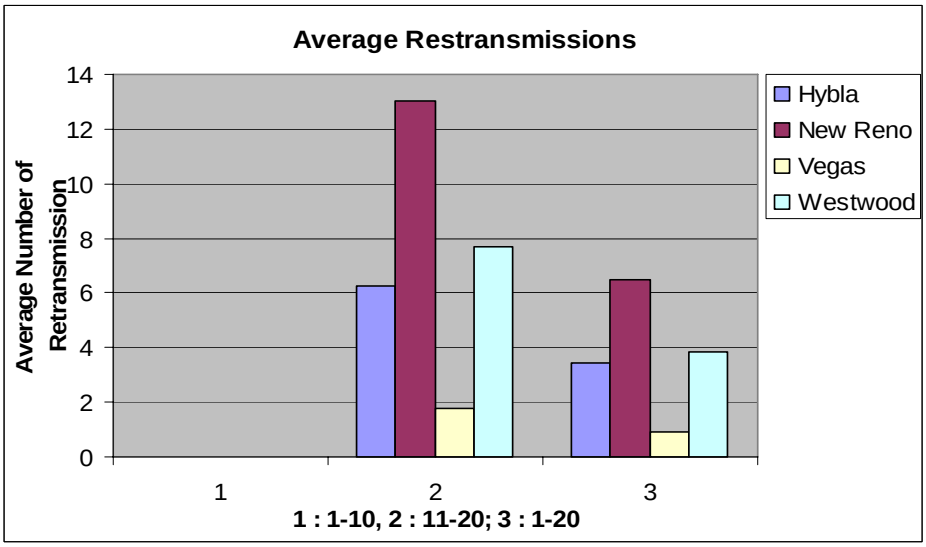
The main performance degrading phenomenon is retransmissions. Retransmissions are either caused by corruptions, congestion or to long delay causing timeouts. Corruption occurs if the CRC check fails, while congestion occurs if the buffers in network routers are overloaded and packets are dropped. Timeout occurs if a TCP sender for some reason does not receive an ACK for a packet. TCP senders have a timer which start when a segment is sent, if and ACK message for the specific segment is not received before the timer expires, the segment is retransmitted. In all these cases a retransmission of the missing packet is necessary. Even though there may be various reasons why the packets fail to arrive, the TCP protocols interpret this as congestion in the network. And since it assumes the same cause in all cases, it also acts in the same way for all retransmissions. In case of congestion, the appropriate thing way to act is to reduce the sending rate, to avoid too much packet loss.

On the first test day no retransmissions were registered, while on the second test day we experienced an average of 7.2 retransmissions per trial. 13 trials experienced retransmissions, which corresponds to 16.25% of the total number of trials, for these trials the average number of retransmission were 22.62. Graph 7 shows an overview of the number of retransmissions for the different trials.



Graph 7 : Number of Retransmitted Segments

Graph 8 shows the average number of retransmissions, test day one and two corresponds to the columns in area 1 and 2, the columns in area 3 is the number average retransmissions for all trials. Since no retransmissions were experienced on the first day and they both contains the same amount of trials, the total average is the half of the average experienced on day two.

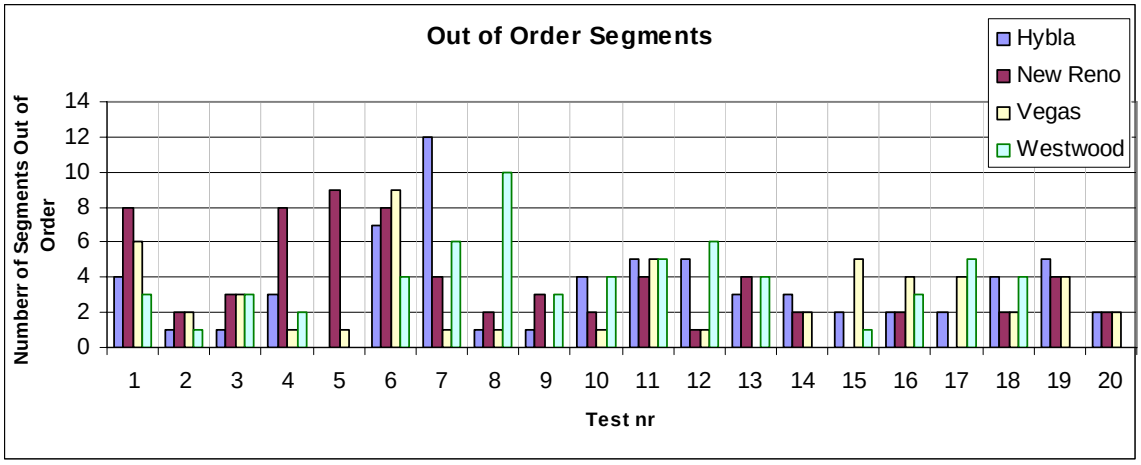


Graph 8 : Average Number of Retransmissions

In our tests the corruption is present because of link degradation, while the congestion is exclusively caused by the single TCP connection. TCP NewReno sets the size of CWND (sending rate) according to the number of acknowledged segments. CWND is increased for every ACK segment, and not decreased until an ACK fail to arrive. It is therefore expected that TCP NewReno will experience more retransmissions than the other TCP variants. This expectation correlates to our findings. As Graph 8 illustrates, TCP NewReno has by far the most retransmissions. The other TCP version uses bandwidth estimations to adjust the CWND. This has some major benefits. Since both TCP NewReno and TCP Westwood+ halves the CWND when duplicate ACKs are received will this have a major degradation of the performance. By estimating the bandwidth TCP Westwood is able to decrease the CWND before segments are dropped, and in this way reduce the retransmissions and the number of times CWND is halved. TCP Vegas has by far the least amount of retransmission. TCP Vegas calculates the difference between estimated output and measured output to avoid

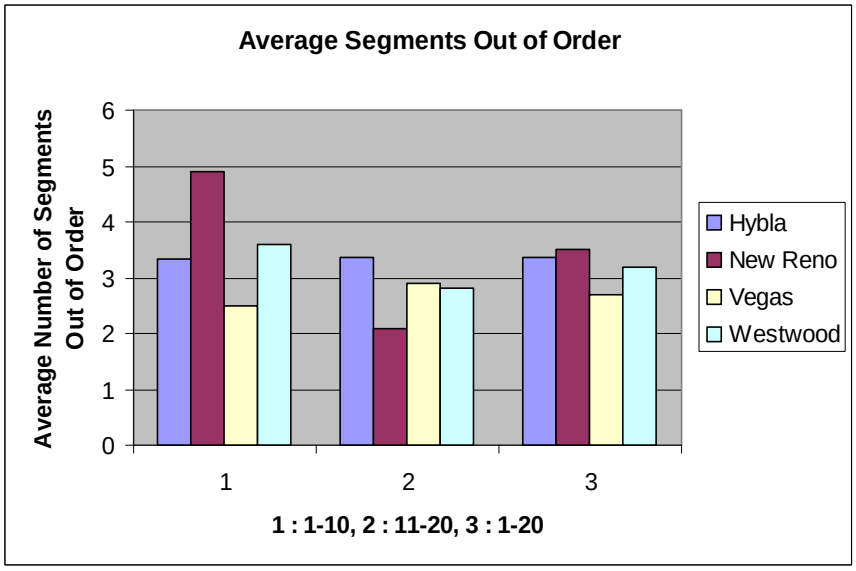
congestion. This has shown to be a very good measurement and TCP Vegas is able to reduce packet loss due to congestion to almost 0. We assume that the retransmission for TCP Vegas is caused by link degradation rather than congestion. TCP Hybla is most different from the other TCP variants, because it only uses bandwidth estimation to increase and decrease the CWND. To be able to do this it has introduced many set of rules. This TCP variant is developed for networks with long RTT and is said to be more efficient then the TCP standard [6] for satellite network, thus a larger CWND has to be expected, as a result losses will be more frequent. In our tests, the loss rate for TCP Hybla is approximately the same as for TCP Westwood+.

The average number of out of order segments was almost the same for the two trial days, 3.6 and 2.8 for trial day one and trial day two, respectively, this gave a total average of 3.2. However, on trial day one they seemed to have a total random distribution, while on the second day they were closer to an average and seemed to be distributed more like uniform distribution. Graph 9 shows all the trials for each TCP variants with the corresponding number of out of order segments.



Graph 9 : Out of Order Segments

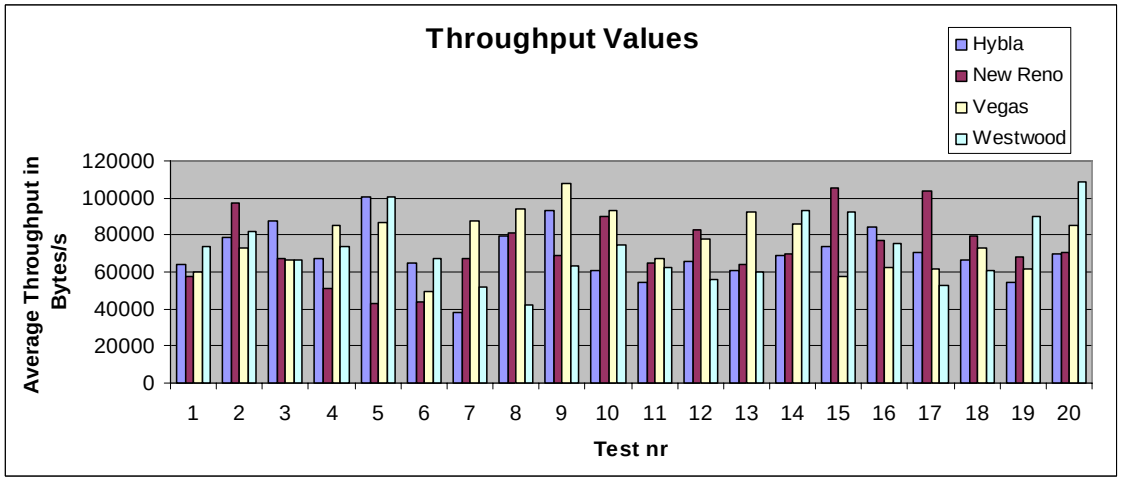
Graph 10 shows the average number of out of order segments. First is the average number of out of order segments for trial day one presented (1), the trial day two (2) and then last the total average.



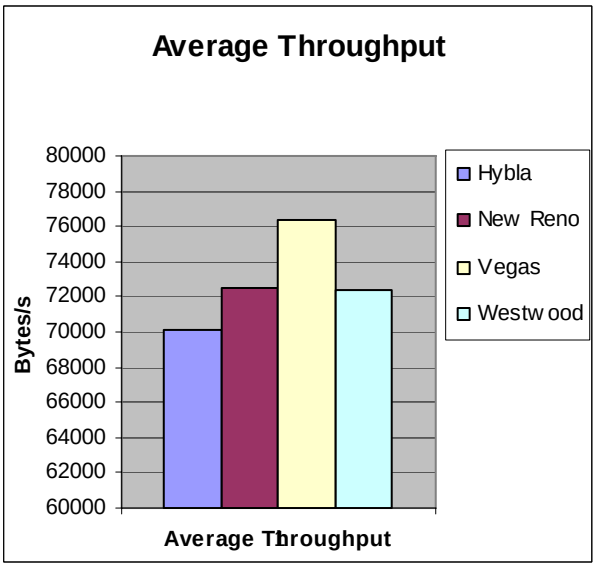
Graph 10 : Average Out of Order Segments

7.1.2 Performance

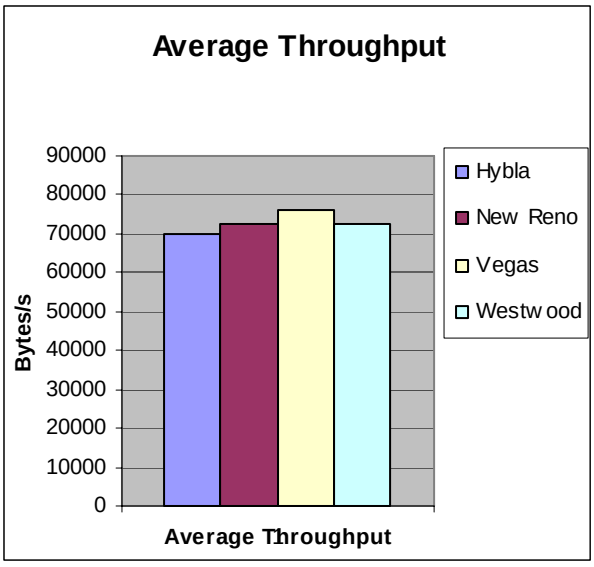
As mentioned, throughput is the main parameter we will evaluate. Graph 11 illustrates the average throughput in bytes for every trial for each TCP variant. Graph 12 and Graph 13 shows the average throughput values for each TCP variant. Graph 12 and Graph 13 differs only in the scale of the Y axis. The values are the average of all 20 trials.



Graph 11 : Throughput Values for Satellite

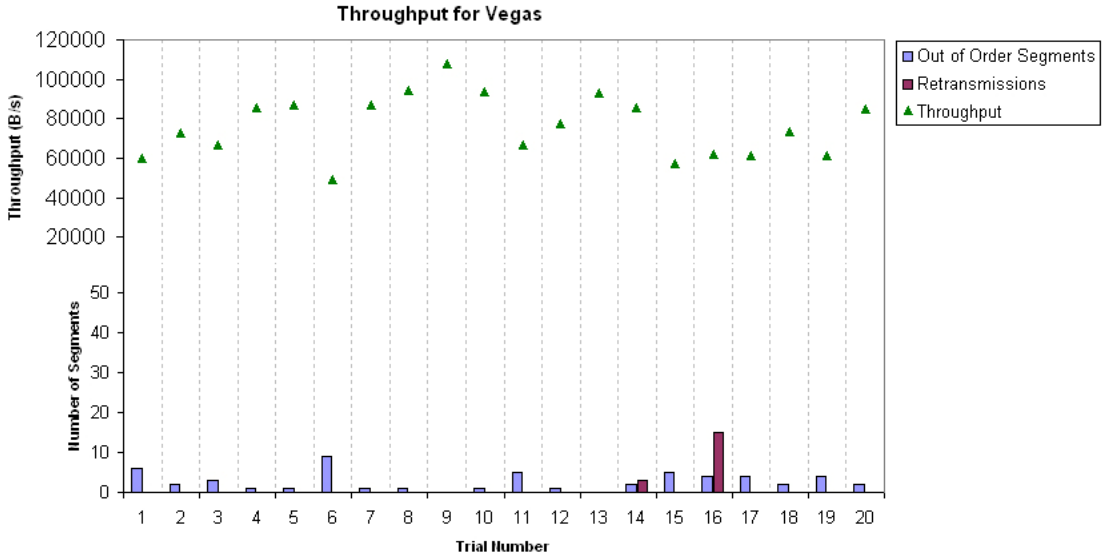


Graph 13 : Average Throughput for Satellite 2



Graph 12 : Average Throughput for Satellite 1

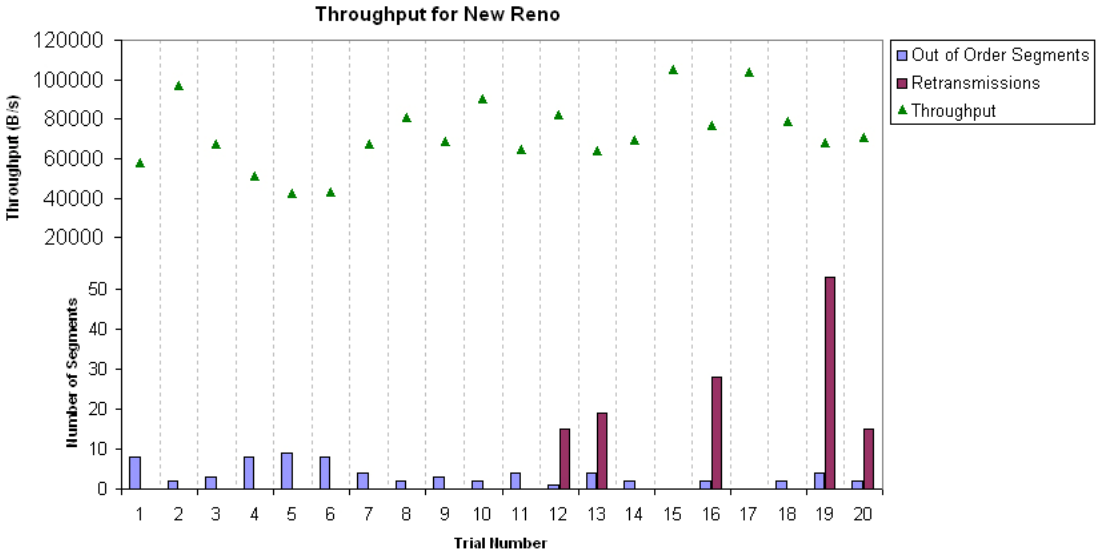
As can be seen; TCP Vegas performed best with an average throughput of 75045.2 B/s, with the minimum of 49347.6 B/s and the maximum of 107535.3 B/s. As mentioned earlier, uses TCP Vegas measures of RTT to estimate the bandwidth, and has linear increase and decrease of the sending rate. This way TCP Vegas is able to find the available bandwidth and continuously utilize it. This results in generally high performance and the best average throughput in our test. Graph 14 illustrates the throughput of TCP Vegas, considering out of order segments and retransmissions. Each trial is shown with both retransmissions and out of order segments. The green triangles in the upper part of the graph illustrate the throughput performance in bytes for the corresponding trials.



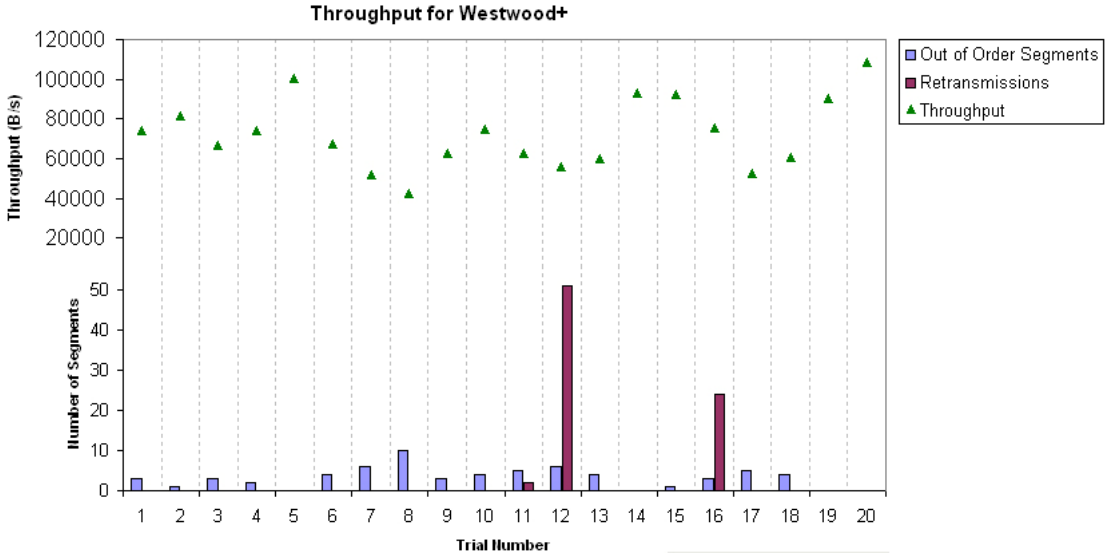
Graph 14 : Throughput for Vegas over Satellite

As seen in Graph 12 and Graph 13 TCP New and TCP Westwood+ have almost equal average throughput. These TCP variants are the two most similar of the test subjects.

The main difference between the TCP variants is the upgrade of TCP NewRenos Fast Recovery algorithm, to the Faster Recovery algorithm for TCP Westwood+. An updated version of TCP Westwood+ takes also advantage of the Adaptive Start (ASTART) algorithm. Prior work has shown that TCP Westwood+ is better to estimate the available bandwidth and this improves the throughput [12]. However, in our test the two TCP variants had a very similar average throughput performance. Graph 15 and Graph 16 shows the throughput performance of the TCP NewReno and TCP Westwood+, respectively. As in graph 14, also the retransmissions and out of order segments are illustrated for each trial.



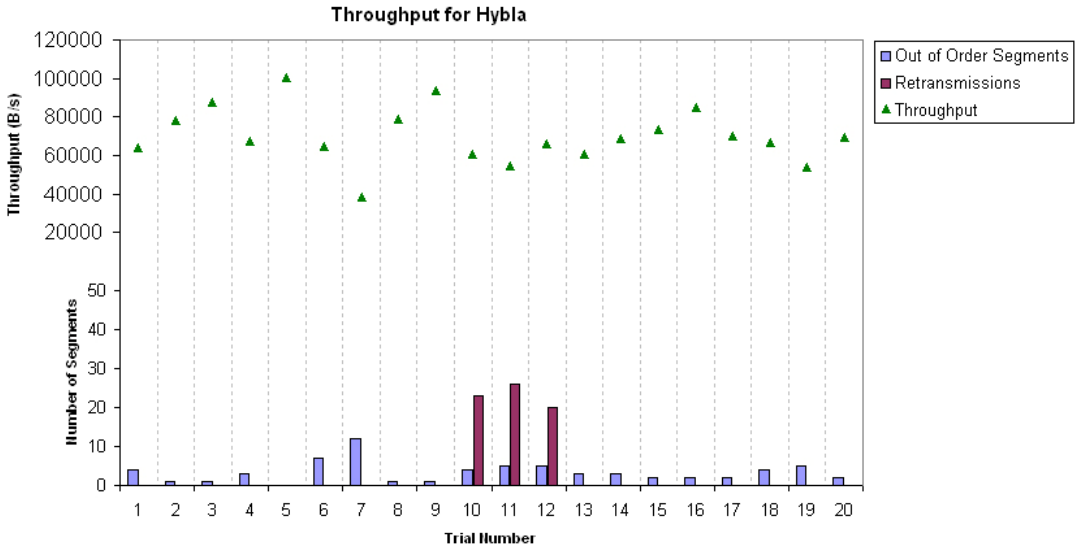
Graph 15 : Throughput for NewReno over Satellite



Graph 16 : Throughput for Westwood+ over Satellite

TCP Hybla had the lowest average throughput performance. This TCP version is created especially for long RTT connections, as is typical for satellite. Prior work [6] concludes that TCP Hybla performs better than TCP NewReno for a variety of RTT. The result of our test concerning this TCP variant was not expected. Graph 17

illustrates the throughput performance of TCP Hybla with the retransmissions and out of order segments in the bottom of the graph.



Graph 17 : Throughput for Hybla over Satellite

If we evaluate the throughput with respect to the out of order segments for the four last graphs, we can easily see that the out of order segments seem to affect the throughput. Out of order segments are normally only present when a TCP connection uses more than one path from the sender to the receiver. If one of the paths has a longer delay than the other, the packets will not arrive in the same order as they were sent. However, in our case the packets had only one way through the network, so no second path could cause delays. Also, there is the question of why the out of order segments should cause that much degradation in the first place? As mentioned, does not the TCP protocol reduce the transmission rate because of out of order segments. Further investigation of the TCP dump files revealed a relative high number of spurious retransmissions. This was revealed by a high number of D-SACK acknowledgments segments and unexpected behaviour. After analyzing the Tcpdump files, we saw three events which occurred more then expected. First was a normal out of order segment situation, caused by three duplicate ACKs. This can be considered as a normal retransmission, and congestion avoidance was invoked. In the second event the sender also had to retransmit because of three duplicate ACKs . In this case the packet was correctly received, and the receiver replied with a D-SACK segment. Thus the sender did not invoke the congestion avoidance. The third event was an out of order segment, caused by time out. Then the sender retransmitted the segment, and waits for an ACK. Then it reveries a D-SACK segment, and instead of setting the CWND to 1 and invoke slow start as it normally does when timeout is experienced, it continues to use the old CWND. Notice that only the two last events cause the generation of a D-SACK acknowledgment. In the cases where D-SACK is sent, the CWND is not increased for one RTT, with many spurious retransmissions, this will be performance degrading. In the cases where a lot of out for order segments were registered, there was also registered many spurious retransmissions, this is why out of order segments seemed to have an effect on the throughput performance.

The spurious retransmissions were caused by timeouts. There are two reasons why this can occur. One reason could be large buffer queues which add extra delay. This

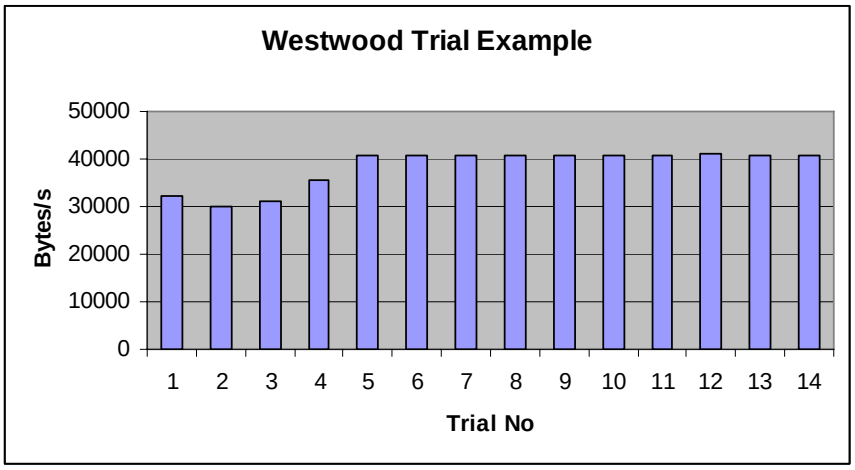
delay, may in addition to the already existing delay in the satellite network, could cause the timer to expire, which would cause timeout and retransmission. Another reason for the timeouts can be delays in the resource allocation. The resource allocation algorithms are constantly analyzing the demanded bandwidth and change the TDMA slots accordingly. The two link end-points negotiate the allocation of the TDMA slots every RTT. This negotiation may cause delays of up to a half RTT. This delay may, in addition to the other delays in the satellite network, cause the TCP senders timer to expire and cause timeouts. To investigate buffer capacity and resource allocation algorithms is beyond the scope of this project, so no further investigation were done to find the exact cause of the spurious retransmissions.

It should also be noted that other research groups that tested during the same period also experienced unexpected behaviour on the satellite test bed.

7.2 Emulation of Ad Hoc Network

A huge amount of data was collected during the emulation of the ad hoc network. The trials varied both in number and in duration. In some of the trials the variation in throughput was large, in these cases we had to increase the number of trials to get the distribution of the throughput values in a normal distribution, with as low standard deviation as possible. A high number of trials will reduce the influence of divergent throughput values. The increase in number of trials will cause the mean value to be closer to the correct average value. In the cases of low packet loss, the variance of the throughput values were small, in these cases we did not run as many trials. As test data we sent 1 M byte of randomly generated data, due to the different throughput values the duration of the trials varied.

During testing we encountered two problems with the emulator. One problem was caused by the random generator in the implementation of the two state Markov model. The first trials always came out with divergent throughput values. The implemented random generator works as follows: one seed is generated, upon generation the second seed, the prior seed is used. This will give the second value a more random pattern. The third seed will again be calculated on the basis the second seed and so on. The random generator needed a few trials two give a random distribution of seed which was acceptable to use. Graph 18 illustrates an example of a trial using Westwood, where each trial is given in Bytes/s. We can clearly see that the throughput values get stable after four trials. If we did not know the cause of the divergent values, we would have to increase the number of trial to get a correct average, as described a bow. Since we knew the cause and the pattern was clear and easily identifiable, we removed these divergent values from the average performance results.



Graph 18 : Westwood Trial Example

The second problem was that the emulator added an extra delay in some cases where the delay was 10 ms. The emulator was created for the primal goal of emulating a satellite network with delays ranging from 500 to 700 ms, and delays as low as 10 ms had never been tested. The extra delay was only added in the case of 10 ms delay. The emulator printed warnings in the output window, so this was generally easy to pick up. These trials were neglected on the basis of the warning messages in from the emulator. However, in some of the trials with 10 ms delay we have an unnatural low throughput performance compared to the 30 ms and the 50 ms trial, this may be cause by these extra added delays.

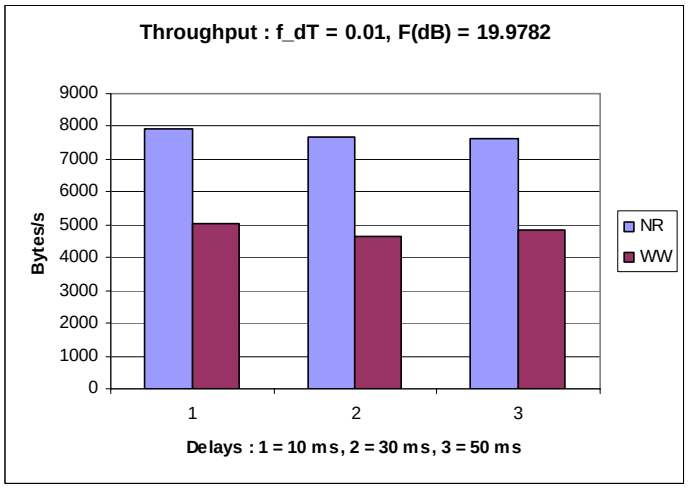
7.2.1 Link Degradation

As mentioned, is the two state Markov model used to simulate the channel. This model introduced two transition variables p_{GG} and p_{BB}. These parameters describe the state of the link at all times, and decide if transmission is possible or not. These are the only parameters which causes link degradation.

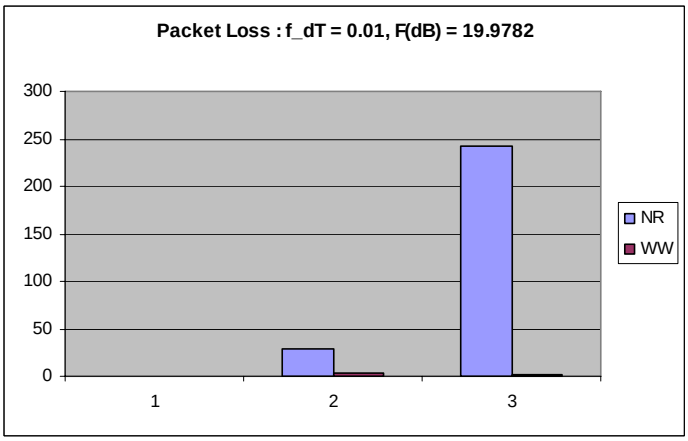
The introduction of delays aggravates the probability of errors. The delay itself is not degrading. The aggravation occurs because the states in the two state Markov model changes over time, not for each transmitted packets. The segments might be sent in a good state, but during the transmission period the model might change from a good state to a bad state. For segments with long delays the probability that the model will change from good to bad state during the transmission is higher.

As previously described, the traffic registration with Tcpdumt took place on the receiver side. The Graph 19 and Graph 20 illustrate the back draw of registering traffic at the receiver, more inaccurate packet loss registration. Graph 20 is a trial where the p_{GG} and p_{BB} is 0.9975 and 0.7543, respectively. This can be interpereted as being a ling with bursty errors, because the probabilities of staying in the current states are high. The expected values probability of packet drop (p_E) is equal to 0.01. And since a total amount of 26215 unique packets are sent, the expected amount of packet loss in 26.2 packets. As can be seen in Graph 20, the number of registered packet loss is close to 250 packets. As can be seen in Graph 19, the throughput is unaffected by the high packet loss, and has a total unaffected throughput values. These two reasons indicate that the packet loss value is wrong. For this reason we

have been very careful before drawing conclusions between the registered packet loss and the throughput values.



Graph 19 : Throughput Example



Graph 20 : Example of Retransmissions

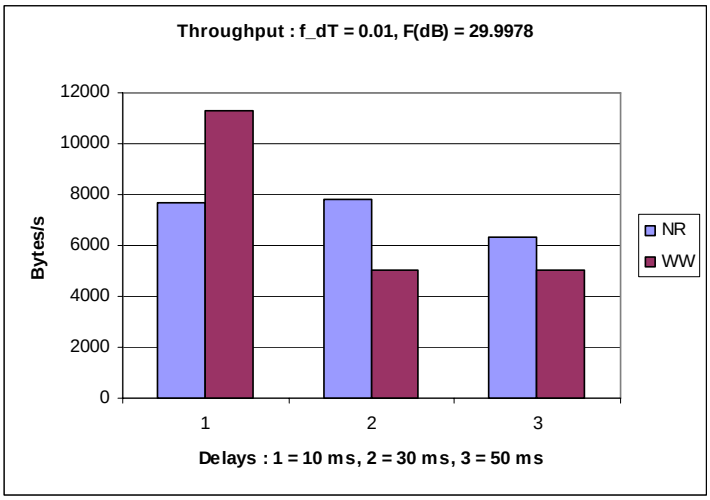
7.2.2 Performance

The throughput is the key factor for the performance evaluation. All the throughput values are given in bytes. The graphs in this subsection illustrate the average performance of TCP NewReno and TCP Westwood+ for the different values of p_{GG} and p_{BB} given in Table 5.

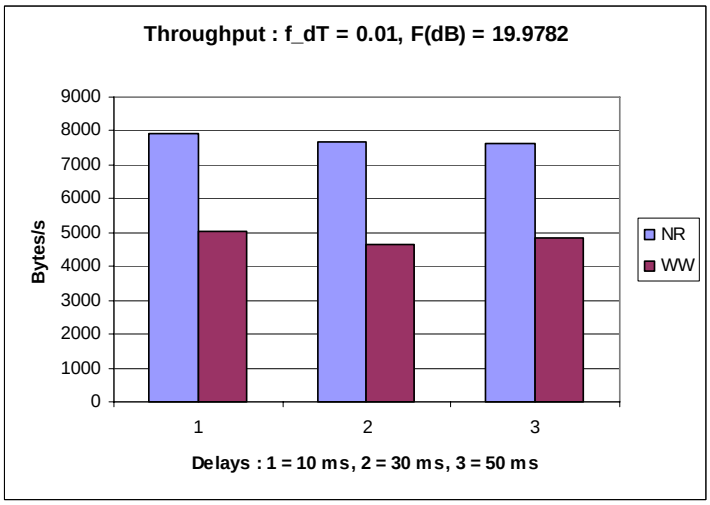
For the three different delays, it is expected that the throughput performance for both TCP variants will decrease when the delay increase. Both variants use ACKs to increase the CWND. If the delay is large, it will take more time for the segments to arrive, and the CWND will increase slowly. On the other hand, with short delays the segments use less time to arrive, and the CWND can increase faster. TCP Westwood+ is known to have a better ability to estimate the available bandwidth, and therefore is more capable of utilizing the link. The effect of better bandwidth estimation is most present of other transmissions and in bottleneck links.

We start at trials with the lowest normalized Doppler bandwidth, f_dT = 0.01, that is from the top of Table 5. The fading margin has the following three values: 29.9978,

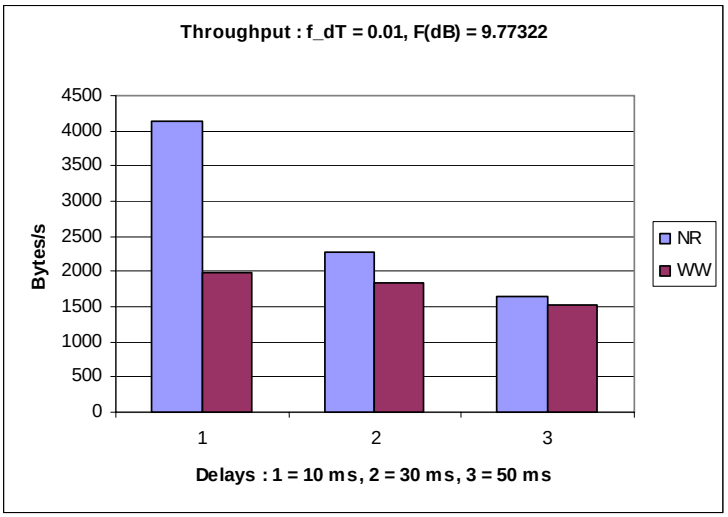
19.9782 and 9.77. For these values we have the corresponding p_{GG} values 0.999329, 0.997518 and 0.991872. As mentioned, do the p_{GG} values describe the probability of staying in the good state. In these cases the probability of staying in a good state is 99.93, 99.75 and 99.18 % respectively. This will again say that we have a 0.07, 0.25 and 0.82 % chance of having a transition from a good state to a bad state. The values for p_{BB} is given as 0.329452, 0.754308 and 0.926851 thus, the chance of staying in a bad state is 32.94, 75.43 and 92.68 %. By comparing these values of p_{GG} and p_{BB} with the values given in Table 5, we can see that these values are higher. This means that the probability of staying a in a state is high, both for the good and the bad state. This causes the model to stay in a state for a longer period, and the effect is bursty segment drops. Graph 21, Graph 22 and Graph 23 illustrate how the different TCP variants handle the bursty environment.



Graph 21 : Throughput : f_dT = 0.01, F(dB) = 29.9978



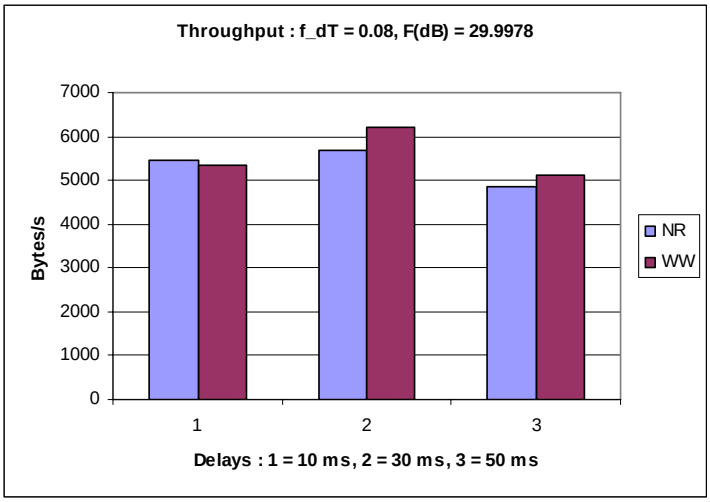
Graph 22 : Throughput : f_dT = 0.01, F(dB) = 19.9782



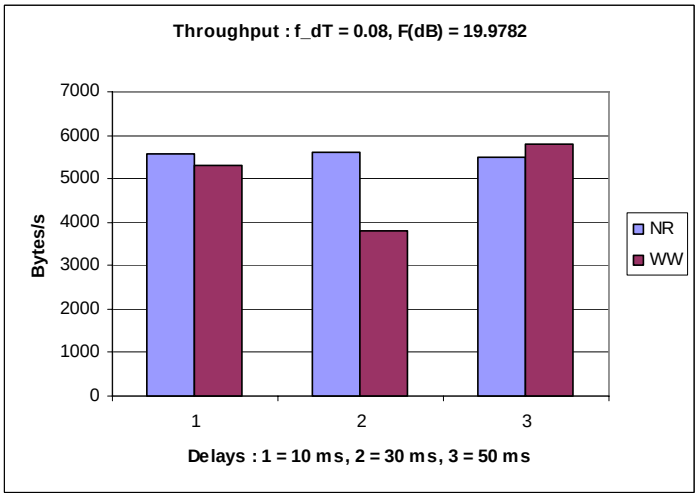
Graph 23 : Throughput : f_dT = 0.01, F(dB) = 9.77322

The general trend in these graphs is that TCP NewReno has a higher average throughput in almost all the cases. The only scenario where TCP Westwood+ performes better is in Graph 21, with 10 ms delay. In this case the probability of packet is low. It seems that in this case Westwood+ is able to take make more use of the available bandwidth compared to TCP NewReno. For the 30 ms and 50 ms trials, the throughput is decreasing as expected when the delay is increasing. This is due to a slower increase in the congestion window. If we look at Graph 22 we can se that both variants have a steady performance throughput event though the delayed is increased. However, the throughput for TCP NewReno is much higher than Westwood+. In this scenario the probability of staying in the bad state has increased to more than 75%, which drastically increase the probability of packet loss. In some cases where the probability of packet loss is high there might be a tradeoff between the high throughput and high packet loss. When the throughput is high, a high number of packet losses will be experienced. If the throughput is reduced, so will the number of packet loss. Because TCP NewReno and TCP Westwood+ react to packet loss by halving the sending rate, the performance is not affected by some decrease in throughput. In Graph 23 the TCP NewReno has a much higher throughput compared to TCP Westwood in the case of 10 ms delay. When the delay increases TCP NewReno is still performing better, but the difference decreases. This last test is the most degrading scenario, with very long duration of the bad state. If the bad state is entered, there is 92.6 % chance that the model stays in this state.

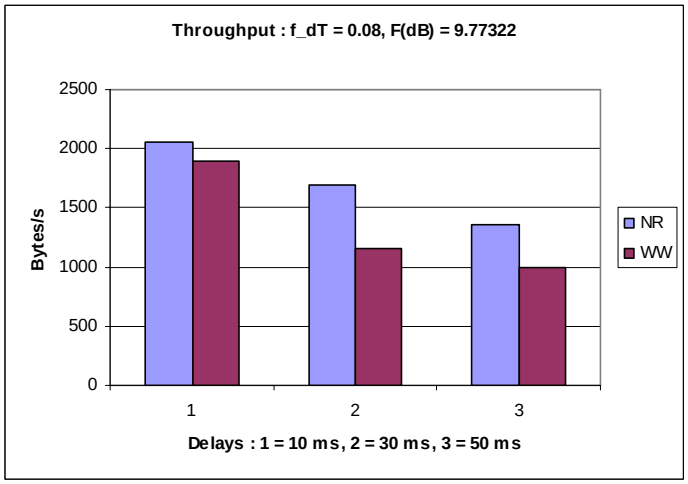
For the second value of normalized Doppler frequency, f_dT = 0.08, the fading margin is as in the previous trials. As we move downward in Table 5, p_{GG} and p_BB are decreasing, which means that the errors will occur less bursty, and more spread. Graph 24, Graph 25 and Graph 26 illustrates how the two TCP variants performed in these scenarios.



Graph 24 : Throughput : f_dT = 0.08, F(dB) = 29.9978



Graph 25 : Throughput : f_dT = 0.08, F(dB) = 19.9782

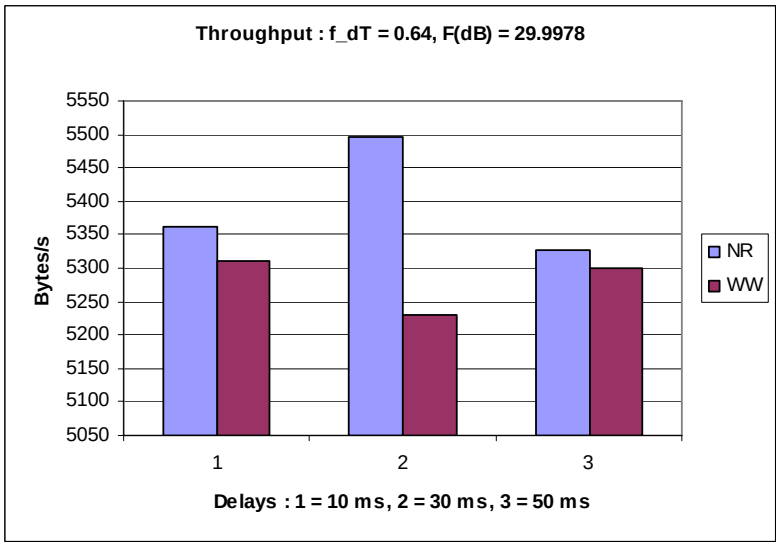


Graph 26 : Throughput : f_dT = 0.08, F(dB) = 9.77322

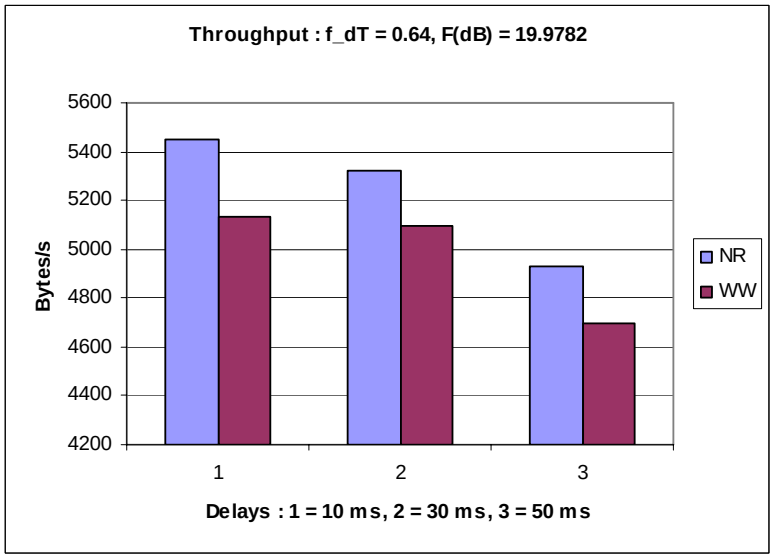
In Graph 24 the throughput values for 10 ms seems to be affected by the added delays described earlier. The 50 ms trials are decreased in an expected ratio compared to the 30 ms delay. These trials have a relative low p_E, the throughput performance shows that TCP Westwood seems to handle this type of traffic better than TCP NewReno.

Graph 25 shows again a trade of situation for TCP NewReno. Graph 26 illustrates a scenario where the p_{E} is large a, here both TCP variants has an expected decrease in throughput as the delay increase. For all trials TCP NewReno has better performance the TCP Westwood+.

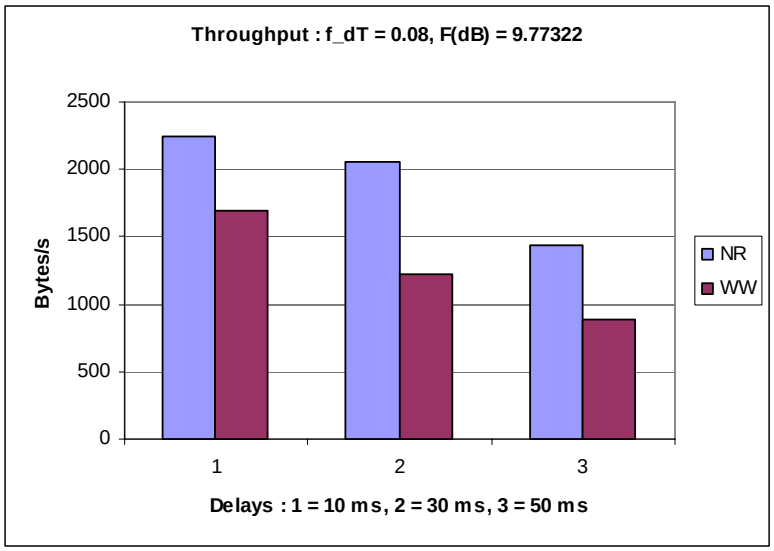
The last series of trials has a normalized Doppler frequency of 0.64, and the same values for the fading margin as the other trials. In these trials the value of p_{GG} and p_{BB} are the lowest compared to the other series of trial. Which means that we experience we experience very short bursts of errors, while still have more the same probability of p_{E} as the prior trials. This results in frequent errors with a short duration. This series are given in Graph 27, Graph 28 and Graph 29.



Graph 27 : Throughput : f_dT = 0.64, F(dB) = 29.9978



Graph 28 : Throughput : f_dT = 0.64, F(dB) = 19.9782



Graph 29 : Throughput : f_dT = 0.64, F(dB) = 9.7732

In Graph 27 seems gain to be affected by the additions of delays within the emulator. Except for this single trial in this single graph, the trend is very clear, TCP NewReno experience higher throughput the Westwood+. This is the case for each of the different delays and for each value of p_{GG} and p_{BB}.

The results we found were unexpected, and differ from prior work. From a theoretical point of view the performance of Westwood+ should be higher. The performance difference might be caused by the two different kernel versions. Different kernels handle TCP variants differently. TCP NewReno used kernel 2.2, this kernel automatically sets the advertised window to half the buffer size, in our case with a buffer size of 16 kB, the advertised window to 8 kB. In kernel 2.2 this value is kept constant for the whole period of the connection. Kernel 2.4 was used on the Westwood TCP sender. This kernel automatically use 4 kB as initial advertised window, but in this case this window is not static, but increases according to the estimated bandwidth. To have a realistic advertised window compared to the buffer capacity, the roof of the advertised window was sat to 8 kB.

8 Conclusions and Further Work

8.1 Conclusion

This thesis has been addressed to the test and comparison of different TCP versions, applied over heterogeneous wireless networks. Since data transmission involves network segment built in different technologies, the aim of this work has been to choose the TCP version which performs best. One solution is to use PEPs, to split the link into smaller links, and use different TCP versions for these minor links. This approach allows using the transport protocol most suited to the specific environment.

For the satellite link we tested four TCP versions: TCP NewReno, TCP Westwood+, TCP Vegas and TCP Hybla. For all versions, we enlarged the TCP buffers on the receiver side to 125 bytes. Thus the maximum achievable throughput is approximately 256 kB/s, which corresponds to a bandwidth of 2Mb/s. For this specific link, we experienced best performance from TCP Vegas. TCP Vegas performed an average throughput of 75045.2 B/s, with the minimum of 49347.6 B/s and the maximum of 107535.3 B/s. The TCP NewReno is an aggressive version, and performed second best throughout the tests. The results of our testing were affected by a high number of spurious retransmissions. This caused in some cases lower throughput performance and affected the behavior of the TCP versions. We expected TCP Hybla to perform better, but based on the results we achieved we can conclude that TCP Vegas is the best TCP version for a satellite link.

We tested two different TCP versions for the wireless ad-hoc network: TCP NewReno and TCP Westwood+. These versions were tested with three different delays, and with nine different conditions of the channel, which results in 27 different test conditions. According to the results we achieved, performed TCP NewReno considerably better in general. We have to take into account the different kernel versions, but based on the results we achieved, we can conclude that TCP NewReno performs better than TCP Westwood+ in wireless Ad-hoc networks, based on the IEEE 802.15.4 standard.

The goal of this thesis was to find an existing TCP version (chosen among the candidates available from the literature), which performs best in overall over heterogeneous wireless networks. Based on the achieved results, and the former conclusions, we can conclude that TCP NewReno probably will perform best through heterogeneous wireless links. In case of using PEPs, we recommend use TCP Vegas on the satellite link, and TCP NewReno in the Ad-hoc network.

8.2 Further Work

We have addressed a very interesting problem, with combining heterogeneous wireless links in networks. For the future testing, we suggest some improvements to be made:

- Modify of the ACE Emulator, to be able to handle short delays. In our case, some of the results were unexpected, because the Emulator added an extra-latency (approximately 2-3 ms) to the delay.

- To be able to get more accurate results, we think it is essential to run several more tests, or modify the random generator implemented on the ACE Emulator.
- Run the tests with the Characteristic Model, and decide the packet drops based on the actual distance between the nodes. It could be of interest to see the difference between the two models.
- Address the problem with spurious retransmissions on the satellite network.

Bibliography

[1] M. Allman, V. Paxson, W. Stevens, “TCP Congestion Control”, RFC 2581, April 1999.

[2] S. Floyd, T. Henderson, A. Gurtov, “The NewReno Modifications to TCP’s Fast Recovery Algorithm”, RFC 3782, April 2004.

[3] C. Casetti, M. Gerla, S. Mascolo, M.Y. Sanadidi, R. Wang, “TCP Westwood: End-to-end Congestion Control for Wired/Wireless Networks”, Wireless Networks Journal, issue 8, 2002, pp. 467-479.

[4] L.S. Brakmo, S. O'Malley, L.L. Peterson, “TCP Vegas: New Techniques for Congestion Detection and Avoidance", Computer Communication Review, Vol. 24, No. 4, pp. 24-35, Oct. 1994.

[5] I. F. Akyildiz, G. Morabito, S. Palazzo, “TCP-Peach: A New Congestion for Control Scheme Satellite IP Networks”, IEEE/ACM Transactions on Networking, vol. 9, no. 3, June 2001.

[6] C. Caini, R. Firrincieli, M. Marchese, T. de Cola, M. Luglio, C. Roseti, N. Celandroni, F. Potorti, “Transport Layer Protocols and Architectures for Advanced Satellite Networks”.

[7] M. Allman, D. Glover, L. Sanches, “Enhancing TCP Over Satellite Channels using Standard Mechanisms”, RFC 2488, January 1999.

[8] M. Mathis, J. Mahdavi, S. Floyd, A. Romanow, “TCP Selective Acknowledgment Options”, RFC 2018, 1996.

[9] J. Border, M. Kojo, J. Griner, G. Montenegro, Z. Shelby, ”Performance Enhanced Proxies Intended to Mitigate Link-Related Degradations”, RFC 3135, June 2001.

[10] M. Allman, S. Dawkins, D. Glover, J. Griner, D. Tran, T. Henderson, J. Heideman, J. Touch, H. Kruse, S. Osterman, K. Scott, J. Semke, “Ongoing TCP Research Related To Satellites”, RFC 2760, February 2000.

[11] J. Mogul, S. Deering, “Path MTU Discovery”, RFC 1191, November 1990.

[12] L.A. Grieco, S Mascolo, “Performance Comparison of Reno, Vegas and Westwood+ TCP Congestion Control” Tech. Rep. N 07 / 2003 / S.

[13] M. Zorzi, A. Chockalingam, R.R. Rao, “Throughput Analysis of TCP on Channels with Memory” IEEE J. Selected Areas Comm., vol. 18, pp. 1289--1300, July 2000.

[14] W.T.H. Woon, T.C. Wan, “Performance Evaluation of Primitive IEEE 802.15.4 for Ad Hoc Wireless Sensor Networks”

[15] CompassRose International Inc, “Introduction to Global Satellite Systems”, http://www.compassroseintl.com/pubs/Intro_to_sats.html, 1999.

[16] A. Ganz, Z. Ganz, K. Wongthavarawat, ”Multimedia Wireless Networks, Technologies, Standards and QoS”, Prentice Hall 2004.

[17] IETF, “Mobile Ad hoc Networks (manet)”, http://www.ietf.org/html.charters/manet-charter.html

[18] ZigBee™ Alliance, “ZigBee Specification v1.0”, 2004

[19] I. Akyildiz, W. Su, Y. Sankarasubramaniam, E. Cayirci, “A Survey on Sensor Networks”, IEEE Commun. May 40 (8) (2002).

[20] W.C Craig, "Zigbee: “Wireless Control That Simply Works” White Paper, 2004.

[21] M. Schwartz, “Mobile Wireless Communication”, 2005

[22] 802.15 Working Group, “IEEE 802.15.4 Specifications: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-

Rate Wireless Personal Area Networks (LR-WPANs)", IEEE Computer Society, 2003
[23] M. del Rey, "Transmission Control Protocol", RFC 793, September 1981.
[24] W. Stevens, "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms", RFC 2001, January 1997.
[25] "HOT BIRD™ 6 :: 13° East", Eutelsat, http://www.eutelsat.com/satellites/13ehb6.html
[26] Skyplex Data Terminal Specifications, 2004 http://votos.isti.cnr.it/files/skyplex.pdf
[27] T. de Cola, M. Marchese, G. Portomauro, "Design and Performance Evaluation of a Packet-Switching Satellite Emulator".
[28] "Skyplex", 2005, http://votos.isti.cnr.it/skyplex.htm
[29] T. de Cola, M. Marchese, M. Perrando, G. Portomauro, "A Packet-Switching Satellite Emulator: A Proposal about Architecture and Implementation".
[30] T. Camp, J. Boleng, V, Davies, "A Survey of Mobility Models for Ad Hoc Network Research", September 2002
[31] S. PalChaudhuri, J.Y. Le Boudec, M. Vojnovi´c, "Perfect Simulations for Random Trip Mobility Models"
[32] M. Chiani, E. Milani, R. Verdone, "A Semi-Analytical Approach for Performance Evaluation of TCP-IP Based Mobile Radio Links", 2000
[33] M. Zorzi, R.R. Rao, and L.B. Milstein, "On the accuracy of a first-order Markov Model for data transmission on fading channels" Proc. IEEE ICUPC'95, pp. 211-215, November 1995.
[34] S. Floyd, J. Mahdavi, M. Mathis, U. Pododolsky, "An Extension to the Selective Acknowledgement (SACK) Option for TCP", RFC 2883, July 2000.

Appendix

Included on the appendix-disc is the source code for ACE system and TCPDUMP files. The ACE folder includes C/C++ code for both Two State Markov Model and the Characteristic Model. The first model is ACE, and the second model is ACENEW. The TCPDUMP files are separated in two folders, one for the satellite link, and one for the wireless ad hoc network.